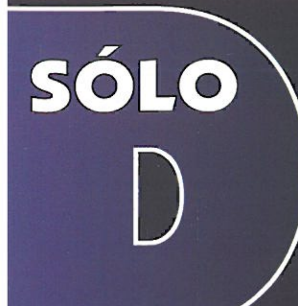
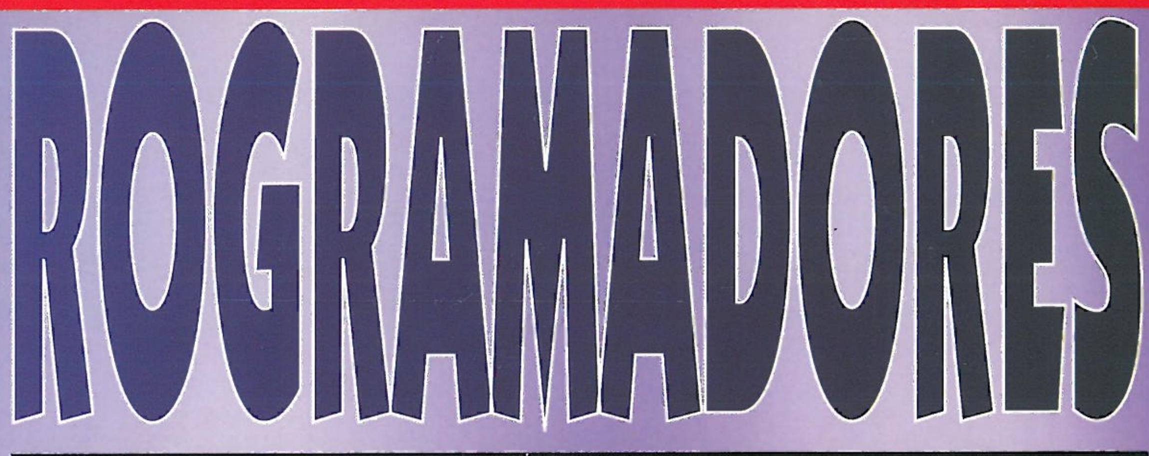
LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO







Revista especializada para usuarios de PC

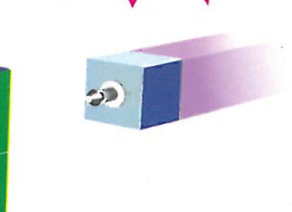
AÑO 3. Nº 23
250 PTAS.
ARGENTINA 9'50 $
CHILE 3000 $
PORTUGAL 1500$

EL PODER DE LOS 32 BITS

WATCOM C++

ESTUDIO DE LOS APPLETS DE JAVA

LINUX: CÓMO CREAR DISCOS DE EMERGENCIA A MEDIDA



Y además:

- Redes locales: interconexión y segmentación
- Grandes sistemas: Objeto natural
- RedesTCP/IP bajo UNIX

CURSOS PRÁCTICOS DE:

- Creación de un compilador
- Visual Basic 4.0
- Visual C++ 4.0
- Programación de una demo







JULIO 1996. Número 23
SÓLO PROGRAMADORES es una publicación de Tower Communications

Editor
Antonio M. Ferrer Abelló
Director Editorial
Mario de Luis García
Subdirector Editorial
Carlos Doral Pérez
Director Financiero
Francisco García Díaz de Liaño
Director de Producción
Carlos Peropadre
Director de Distribución
Enrique Cabezas García
Directora Comercial
Carmina Ferrer

Director
Mario de Luis García
Redactor Jefe
Carlos Doral Pérez
Coordinador Técnico
Eduardo Toribio Pazos
Colaboradores
Fernando de la Villa, Fernando J. Echevarrieta, Pedro Antón. Juan M. Martín, Luis Martín, José M. Peco, José C. Remiro, Agustín Guillén, Daniel Navarro, Jorge del Río, Enrique De Alarcón, David Aparicio, M. Jesús Recio, Santiago Romero, Miguel Cubas, Vicente Cubas

Jefe de Diseño
Fernando García Santamaría
Maquetación
Clara Francés
Tratamiento de Imagen
María Arce Giménez
Servicios Informáticos
Digital Dreams Multimedia, S.L.
Secretaría de Redacción
Rosa Arroyo
Suscripciones
Erika de la Riva
Redacción, Publicidad y Administración
C/ Aragoneses, 7
28100 Pol. Ind. Alcobendas (MADRID)
Telf.: (91) 661 42 11 / Fax: (91) 661 43 86
Filmación
Filma Dos S.L.
Impresión
Gráficas Reunidas, S. A. Madrid
Distribución
SGEL

Depósito legal: M-26827-1994
ISSN: 1134-4792

EDITORIAL

UNIVERSIDAD, TIRAMOS LA PIEDRA Y ...

Se comentó en un anterior número, en esta misma editorial, sobre el gran problema de la proliferación de academias de informática, sigamos ahora que ya han acabado las clases, el análisis subiendo unos escalones.

Muchos intelectuales piensan y no sin razón, que el futuro de un país se teje en la Universidad. Es por tanto de vital importancia adaptar las enseñanzas impartidas en tan importante Institución a la realidad cotidiana que demanda el mercado laboral. No sé exactamente si esto se ajusta en carreras como Medicina, Pedagogía, Periodismo o Arquitectura, pero me temo que la respuesta es no en algunas Escuelas y Facultades de Informática de nuestra país. Pero lo fácil es señalar esto, y seguro que mucha gente no está de acuerdo con esta afirmación, lo difícil es analizar el problema y sus causas:

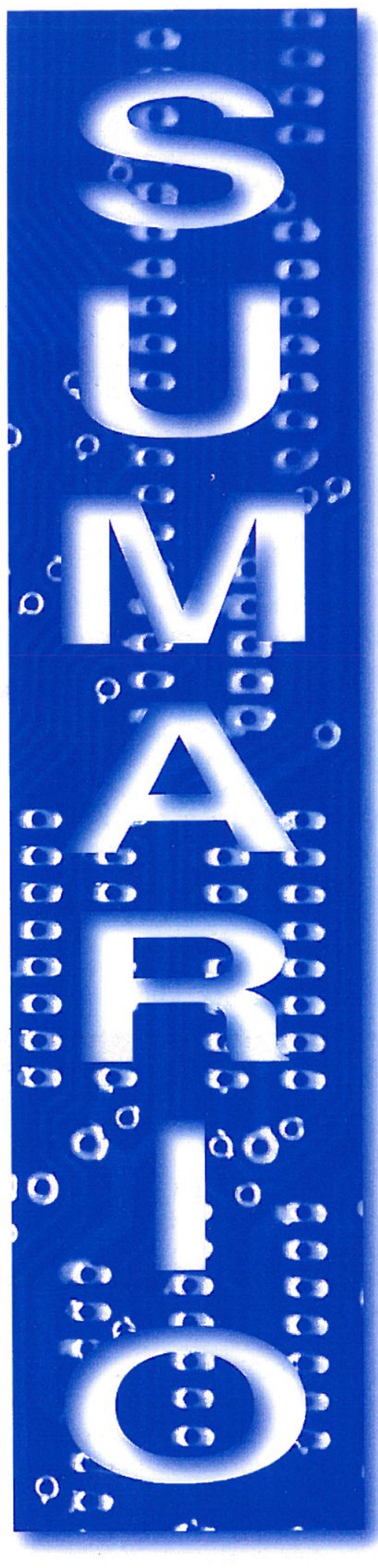
Se podría argumentar lo cambiante de las tecnologías relacionadas con nuestro sector, o los presupuestos ajustadísimos, es cierto, pero yo encuentro un primer problema en la calidad de la docencia en nuestras universidades. Esta afirmación acarrea injusticia, pues se está generalizando, existen grandes profesores en la Universidad, pero lamentablemente existen otros muchos bastante malos. Si se consiguiese realizar un buen filtrado, los beneficiados seríamos todos.

La Universidad con sus áreas de I+D debería ir por delante de la demanda laboral y no al revés. Conocemos en la redacción de la revista muchos estudiantes, grandes programadores que superan en nivel de conocimientos a sus "maestros", son programadores que están al día en las más recientes tecnologías, saben que este o aquel lenguaje va a ser normalizado por primera vez en un par de meses en no sé cual reunión de San Francisco. No es que sepan algo más, sino que saben bastante más.

La Universidad lo tiene difícil, ha de competir con el sector privado en busca de los mejores profesionales, y una vez encontrados, algo más difícil, que estas personas sean pedagógicamente aptos. Quizá sea ese otro de los problemas, no todos los buenos programadores son potencialmente buenos transmisores de sus conocimientos.

Una vez que estuviese salvado este problema, entraríamos en el segundo, los temarios a tratar en las diferentes asignaturas. Alguien que salga de la Facultad o la Escuela de Informática y no se haya complementado su formación como programador por sí mismo, tiene el potencial de ser un buen programador y en el futuro analista o jefe de proyecto, pero no es al momento de salir de la Universidad un gran programador. La solución no es fácil de encontrar. Los seminarios, bueno, es necesario que se impartan seminarios de aquello que más demanda el mercado, me parece ridículo un licenciado en informática en una academia de Visual Basic, y eso se está dando. Seminarios interesantes serían por ejemplo: Lenguaje HTML, Java, Programación gráfica, Herramientas visuales como Visual Basic y Visual C++, sistemas operativos Linux, OS/2, grandes sistemas etc. Sabemos que en determinadas Escuelas y Facultades tratan de subsanar este problema incluyendo estos seminarios optativos de los temas anteriormente citados dentro de los temarios, pero en la mayoría de los casos son demasiado breves y poco profundos por lo que no hacen nada más que dar una mano de pintura a tan gris panorama.

Sé que hay muchos profesores de Universidad que hablan con nosotros que suscriben estas líneas, pero se encuentran maniatados por diversas causas que no vienen ahora a colación. No deseo una mala interpretación de estas líneas, hay muchos buenos profesores, como hay buenos y malos médicos, o buenos y malos periodistas, simplemente que en la Universidad sería especialmente deseable que todos los profesores fueran buenos, y es algo que se podría conseguir. Nosotros simplemente tiramos la piedra...


SUMARIO


23

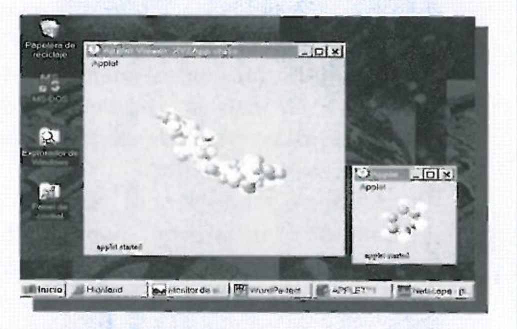
WATCOM C++. (Pág. 34)



EL PODER DE LOS 32 BITS

Watcom C++ es el compilador favorito de muchos desarrolladores de software, especialmente es utilizado por los programadores de video-juegos, gracias a su versatilidad y su fácil manejo de la memoria alta del sistema.

TECNOLOGÍA WEB. (Pág. 10)



EL LENGUAJE JAVA

Java puede ser el futuro de la Red, Sólo Programadores quiere que todos sus lectores se familiaricen con este lenguaje, el cual, en poco tiempo será demandado por muchas empresas en sus ofertas de empleo. Muy interesante y de recomendable lectura es, también, el artículo de este mes de la habitual sección de Linux titulado "Discos de emergencia". Además todas nuestras habiituales secciones siguen avanzando con un solo propósito, proporcionar la máxima información de cada área técnica al lector.

Segunda generación de adaptadores Olicom

Olicom ha anunciado el adaptador Token-Ring PCI/II 16/4, una tarjeta de segunda generación que incluye soporte del nuevo estándar "Dedicated Token-Ring" y aumenta de forma considerable las prestaciones de red en los entornos Token-Ring conmutados. Este adaptador proporciona una solución ideal para los servidores con conexiones full duplex y los clientes con conexiones de red dedicadas de ancho de banda completo.

El nuevo adaptador combina una versión mejorada de la tecnología PowerMACH de Olicom y un mejor diseño ASIC para mantener la utilización de la CPU al más bajo nivel. Proporciona instalación Plug-and-Play, y es totalmente compatible con los estándares más destacados, incluyendo la última versión PCI Local Bus Specification 2.1, IEEE 802.5r para Token-Ring dedicado, IEEE 802.2 e IEEE 802.5. También incluye soporte para DMI (Desktop Management Interface) y un agente de sobremesa SNMP.

El adaptador ya está disponible en volumen al PVP recomendado de 325 dólares e incluye una garantía de por vida.

Para más información:
Olicom Ibérica
Jacobo de Cal
Tel: (91) 372 98 14
Fax: (91) 372 96 45

Nueva versión de LANDesk Management Suite

Intel Corporation anunció recientemente una versión enriquecida de su software de administración de red. La versión 2.5 incluye nuevas características como gestión vía Internet y compatibilidad con el entorno Windows NT, que permiten un control mejorado de las redes PC con la arquitectura de Intel y un mayor tiempo de funcionamiento para los usuarios.

LANDesk Management Suite 2.5 dispone de herramientas que simplifican la tarea de añadir nuevos sistemas bajo Windows NT en la red. Incluye una suite de aplicaciones de gestión críticas compatibles con entornos mixtos de servidores NetWare y Windows NT y sus clientes, lo que supone un ahorro de tiempo para el administrador de la red y una mayor productividad.

La nueva versión proporciona una función de administración multi-NOS integrada para las estaciones de trabajo bajo DOS, Windows 3.x y Windows 95 que se conectan a redes NetWare o Windows NT. Esto permite a los administradores controlar una red local multi-NOS desde una única consola central. Además, los administradores de red pueden efectuar operaciones de inventario de parque y de toma de control remoto en estaciones de trabajo IBM OS/2.

LANDesk Management suite 2.5 ya está disponible al precio recomendado de 95.250 pesetas para una licencia de cinco nodos. Existen precios especiales para un mayor número de nodos.

Para más información:
Intel Corporation Iberia
Tel: (91) 308 25 52
Fax: (91) 310 54 60
World Wide Web: http://www.intel.com

Acelerador de rendering para MicroStation 95

Bentley Systems ha lanzado al mercado el software Quick Vision, un acelerador de rendering que se aplica a MicroStation 95. Quick Vision está basado en el archivo Renderware de Criterion Software y está disponible para todos los usuarios de MicroStation 95 que trabajan con Windows.

El nuevo software ofrece unas características de renderización que antes sólo se incluían en el hardware de gráficos especializado. Permite a los usuarios efectuar todos los diseños virtualmente y visualizarlos mientras están en el modo de rendering. El usuario puede hacer girar los objetos de la pantalla, editarlos e incluso asignarles nuevos materiales en tiempo real y sin necesidad de hacer un reajuste en el modo wireframe. Además, las animaciones se pueden construir y previsualizar en tiempo real. Quick Vision incluye un renderizador autónomo que permite a los diseñadores que no trabajan con MicroStation, visualizar los archivos de esta aplicación.

Quick Vision se puede ejecutar en Windows, Windows 95 y Windows NT, aunque está previsto que en breve pueda correr en otras plataformas de MicroStation

Nuevo microcontrolador MCS

Intel Corporation ha anunciado la llegada del 80C151, un nuevo miembro de su familia de microcontroladores MCS de 8 bits. Se trata de un controlador que ofrece unas prestaciones intermedias y un precio situado entre el del dispositivo original 80C51 y el de los microcontroladores de la serie 8xC251.

El 80C151 ofrece compatibilidad drop-in con el controlador 80C51. El nuevo modelo reemplaza la unidad de ejecución de instrucción secuencial del 80C51 por una estructura en pipeline de dos relojes por instrucción, lo que constituye una mejora de 6 veces de tiempo de ciclo.

También incluye algunas de las características del dispositivo avanzado 80C251 que no están disponibles en el 80C51, incluyendo un bus de código interno de 16 bits, modo página, capacidad externa de estado de espera, una serie de controladores programables, un cronómetro watchdog hardware y tres cronómetros/contadores.

El nuevo microcontrolador se adapta de forma adecuada a las aplicaciones con las unidades de CD-ROM, los escáneres, los terminales de punto de venta, las tarjetas de telecomunicaciones, los teléfonos y los modems.

El controlador 8xC151 está disponible en versiones ROM, sin ROM u OTP (One Time Programmable). Los dispositivos se presentarán en encapsulados PDIL (Plastic Dual In Line) de 40 cables, o bien PLCC (Plastic Leaded Chip Carrier) de 44 cables.

Nuevos productos de Bay Networks

Bay Networks ha anunciado nuevos módulos de routing y de conmutación para su hub System 5.000. Estos módulos incluyen el primer módulo integrado de routing de red virtual para la conectividad a redes virtuales existentes y la migración a ATM, además de un módulo de conmutación Ethernet/Fast Ethernet para la conmutación de redes y módulos routers para el rutado escalable a redes de área local (LAN) y redes de área amplia (WAN). El módulo de routing de red virtual ATM en el System 5.000 proporciona conexiones mediante un único interfaz, con una capacidad de proceso de 120.000 paquetes por segundo.

Los módulos routers soportan cuatro interfaces backplane de hubs Ethernet o Token-Ring, y dos módulos de red opcionales Ethernet, Token-Ring, FDDI, 100BASE-T, Sincrónicos y RDSI BRI. Las prestaciones oscilan desde tasas de transporte de 50.000 paquetes por segundo para un sólo módulo hasta los 200.000 paquetes por segundo en una configuración de cuatro módulos.

El nuevo módulo de conmutación Ethernet aumenta sensiblemente el ancho de banda disponible sobre las redes de medios compartidos mediante la segmentación del tráfico en la red para reducir cuellos de botella de datos. Presenta una capacidad interna de conmutación de dos gigabits por segundo y soporta diferentes interfaces en un sólo módulo, permitiendo la elección de conectividad 10 Mbps, 100 Mbps o FDDI.

Para más información: Bay Networks
Ricardo Miranda-Naón
Tel: (91) 383 13 20

Sybase lanza la Conectividad Universal

Sybase ha desvelado la iniciativa Conectividad Universal, para conectar, extender y simplificar grandes entornos informáticos distribuidos y heterogéneos. La iniciativa comprende diversas ampliaciones de la familia de productos de conectividad Sybase Enterprise CONNECT, permitiendo a los clientes acceder, analizar, actualizar y gestionar cualquier dato de la empresa en cualquier arquitectura, plataforma o lugar.

La segunda generación de productos de conectividad Enterprise CONNECT, que ha sido ampliada y mejorada, se encuentra en el núcleo de la iniciativa. Las mejoras de Enterprise CONNECT incluyen, entre otras características, las siguientes:

- Simplificar la administración de entornos distribuidos mediante la facilidad de uso de las herramientas de gestión.
- Extender la escalabilidad, rendimiento y exactitud de las aplicaciones cliente/servidor disponibles hoy en día mediante la implantación de sistemas de enlace múltiple.
- Conectarse a múltiples bases de datos, aplicaciones, plataformas y objetos de las grandes empresas distribuidas.
- Extender los sistemas existentes a nuevas tecnologías a medida que estas van apareciendo en el mercado.

Para más información:
Sybase Iberia S. A.
Marcos Carmena
Tel: (91) 302 09 00
E-mail: marcos.carmena@sybase.com

Cisco y Olicom establecen una alianza estratégica

Cisco Systems y Olicom han anunciado un acuerdo a largo plazo para desarrollar conjuntamente tecnología Token-Ring para productos de conmutación y routers. Según este acuerdo ambas compañías desarrollarán conjuntamente la futura generación de productos de conmutación Token-Ring para proporcionar una conectividad Token-Ring dedicada y económica en entorno de sobremesa. Cisco se encargará de proporcionar la tecnología de conmutación y de software de sistema, mientras que Olicom proporcionará la tecnología Token-Ring a Cisco, incluyendo el software de altas prestaciones PowerMACH. Cada una de ellas comercializará los productos resultantes.

Los productos de conmutación Token-Ring que resultarán del acuerdo con Olicom complementarán las otras capacidades Token-Ring que Cisco ofrece en el mercado de conmutación de redes como parte de sus planes de lanzamiento de productos CiscoBlue.

El acuerdo reforzará la posición de Olicom y ampliará su oferta de productos en el mercado de conmutación de redes de rápido crecimiento.

Para más información:
Olicom Ibérica
Jacobo de Cal
Tel: (91) 372 98 14
Fax: (91) 372 96 45

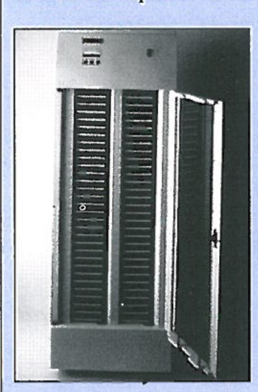
LIBROS

Mini-Librería de cartuchos 545OML para AS\400

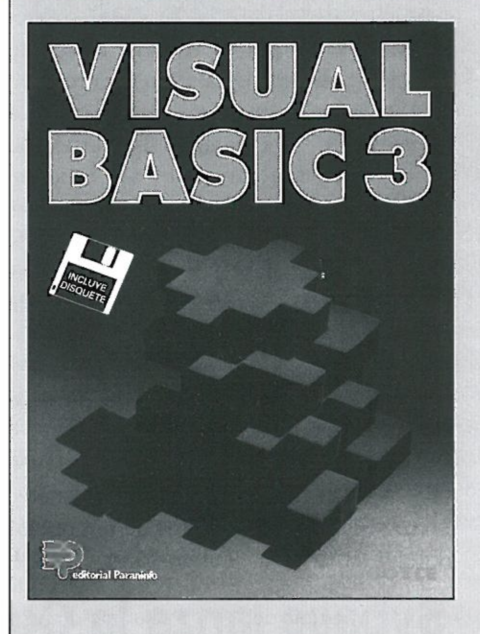
Memorex Telex ha lanzado al mercado la mini-librería de cartuchos de media pulgada 5450ML, diseñada para proporcionar una solución de almacenamiento flexible y de alto rendimiento a bajo coste para entornos AS/400, UNIX y redes de ordenadores PC.

La nueva mini-librería dispone de interfaz SCSI y es una solución de disponibilidad inmediata y fácil implantación con un precio competitivo. Según Carlos Franco, responsable en España de la estrategia y productos de almacenamiento de Memorex Telex en España, "la 5450ML aporta los niveles de velocidad y fiabilidad que nuestros clientes demandan y la compatibilidad con sus inversiones en tecnologías de almacenamiento y gestión de cartuchos, tanto en hardware como en software".

El nuevo producto de Memorex Telex se ofrece a un precio de 3.900.000 pesetas, incluido un sistema completo con una unidad de cartucho de media pulgada y un autocargador de 60 cartuchos. La compañía complementa la mini-librería con una amplia oferta de Servicios Avanzados. Estos servicios incluyen consultoría y planificación estratégica, instalación, integración de configuraciones de almacenamiento, mantenimiento de productos y formación.



Visual Basic 3

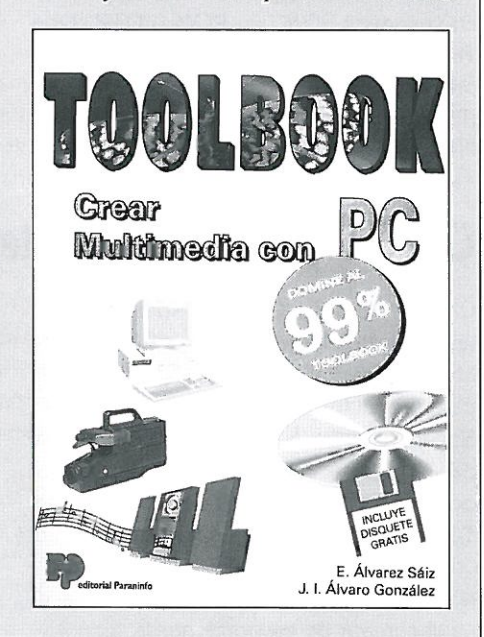


Una de las herramientas de desarrollo para el entorno Windows más conocidas es Visual Basic de Microsoft. Con la ayuda de este libro eminentemente práctico cualquier usuario de ordenadores puede aprender a utilizar la versión 3 de Visual Basic empezando desde cero. Los primeros capítulos están dedicados a las ideas y operaciones básicas, y ayudan al lector a comprender la filosofía de trabajo de la herramienta. Los siguientes capítulos explican la manera de dominar las características más usadas como los menús, las cajas de diálogo y los controles de acceso a archivos. La última parte del libro trata las características avanzadas que permiten crear programas más sofisticados. El libro incluye ejemplos y viene acompañado por dos discos con proyectos de ejemplo y actualizaciones de librerías.

Editorial: Paraninfo
Autor: Jim Boyce
424 páginas
Idioma: Castellano
Precio: 4300 (iva inc.)

Toolbook. Crear Multimedia con PC

Toolbook es una de las múltiples herramientas de creación multimedia que existen en el mercado. La presente obra está pensada para los desarrolladores que desean introducirse en el mundo multimedia con la ayuda de esta aplicación. Se trata de un libro muy didáctico, estructurado de tal forma que el lector va adquiriendo los conocimientos necesarios para dominar la herramienta de forma progresiva y escalonada. Se tocan diversos temas como los objetos en Toolbook, los visores, el depurador de guiones, el lenguaje de programación OpenScript y la herramienta Multimedia Toolbook. Cada bloque del libro contiene diversas prácticas para afianzar los conocimientos que se complementan con los ejemplos incluidos en el disco adjunto.



Editorial: Paraninfo
Autores: E. Álvarez Sáiz y J. I. Álvaro González
456 páginas
Idioma: Castellano
Precio: 4500 (iva inc.)

CARTAS DEL ILUSO



El sistema informático del Hospital Modelo modeliza en su corazón electrónico todo el funcionamiento del complejo, regulando admisiones, altas, urgencias, mantenimiento, guardias, disponibilidad de quirófanos y material. Un sistema estadístico detallado permite a los directores tomar decisiones informadas sobre la base de las más mínimas variaciones en los parámetros de gestión. El tiempo medio de estancia de los pacientes, las dosis de adrenalina que se administran en las paradas cardiacas o la proporción de proteínas en los menús del comedor, se almacenan en los servidores de red del complejo, y se someten una y otra vez a toda clase de comparaciones cruzadas, buscando la forma de optimizar los costes del hospital y la salud de sus pacientes.

Esther era la directora de Sistemas de Información del hospital. Licenciada en Medicina, había estudiado Informática como segunda carrera y había dejado las consultas, volcándose en el diseño del sistema.. Solía decir que el mejor progreso que podía hacer la medicina moderna era mejorar la gestión, que el problema de la salud es hoy en día un problema económico, y que hacer con el dinero y los recursos de un hospital el milagro de los panes y los peces era la forma más eficaz de salvar la vida de la gente. El resto de los médicos no le miraban como un igual: guardaban un desprecio secreto hacia esa compañera que había decidido alejarse de los pacientes. Quizás porque no tenía los arrestos de compartir su sufrimiento.

Pero ella también les despreciaba a ellos, porque a fuerza de acercarse a la gente, y quizá para no volverse locos, habían matado su propia vulnerabilidad, y trataban a los familiares como si fueran pueblerinos ignorantes, patéticos en su convencimiento de que su pequeña tragedia particular era la más importante del mundo, o se paseaban entre las camas con los historiales debajo del brazo y recitando a los alumnos de Prácticas, con voz didáctica y desapegada, las dolencias de los desahuciados.

Esther, mientras tanto, seguía retocando una y otra vez las variables de optimización del sistema, buscando la manera de que las salas de intervenciones estuvieran ocupadas durante menos tiempo para acortar las listas de espera, o intentando descubrir patrones de incidencias para aumentar las guardias por anticipado ante un previsible aumento de los nacimientos en Obstetricia o los politraumatizados en Urgencias.

Un día, Esther frunció el ceño delante de la pantalla, se quitó las gafas, las limpió y se las puso de nuevo. En uno de los histogramas, una ventana titulada "Exitus", la barra más a la derecha no existía. Eso significaba que el número de muertes en el hospital había sido cero en el último mes. Algo estadísticamente imposible en un hospital del tamaño del Hospital Provincial. Hizo revisar las rutinas de visualización, las de cálculo, y finalmente la base de datos. Nadie encontró nada anómalo.

Esa tarde habló con los jefes de departamento. Los datos eran reales. Por alguna razón extraña, ese mes los enfermos terminales seguían agarrados a la vida, sin evolucionar en ningún sentido; las paradas cardíacas en la UCI se resolvían con apuro, pero felizmente; los partos más difíciles salían adelante; los prematuros de menor peso seguían moviento sus diminutos pulmones dentro de las incubadoras. Durante los últimos treinta días, y contra todo pronóstico, la muerte había faltado a su trabajo habitual en el complejo. Todos los que estaban graves seguían sin mejorar, y la angustia de sus familiares crecía, pero ni siquiera las muertes más esperadas, las más convenientes, las que habrían puesto fin a un calvario interminable para los pacientes y sus familias, se habían producido. Era como si alguien hubiera proscrito la defunción. Y esa desviación en los parámetros estaba empezando a perjudicar a las demás variables, disminuyendo el número de camas libres, aumentando la concentración de familias en los pasillos. Esos desahuciados no acababan de comprender que debían mejorar o dejar la cama libre, pero no hacían ni lo uno ni lo otro. Eran como el perro del hortelano.

Pasaron dos semanas más. La muerte, esa compañera despreciada pero imprescindible para el funcionamiento de un hospital, seguía sin cumplir su obligación.

Una mañana, uno de los programadores corrió hacia Esther agitando un listado. Había un error en el sistema de captura de datos: una de las últimas actualizaciones, realizada con la intención de arreglar un pequeño problema en la gestión de fechas, había hecho erróneo el proceso de los decesos. Todas las defunciones actuales se registraban erróneamente en el futuro. Era sólo cuestión de cambiar un signo "menor o igual" por un signo "menor" dentro de un bucle de iteración. Esther autorizó el cambio sin levantar la vista del papel.

A las veintitrés cincuenta, las nuevas correcciones se hicieron efectivos. A las veintitrés cincuenta y uno, un cardiógrafo de la UCI de la cuarta planta emitió un pitido y mostró una línea continua. A las dos dieciocho, una diminuta caja torácica dentro de una incubadora dejó de moverse. A las ocho y siete minutos, unos ojos se cerraron para siempre. Todo estaba correcto. El hospital podía funcionar bien de nuevo.

LOS APPLETS EN JAVA

Fernando J. Echevarrieta

Para realizar una página de WWW "Java Powered", no es necesario saber programar en lenguaje Java. En el presente artículo se estudiará cómo incorporar applets Java predefinidos a documentos HTML, cómo combinarlos con CGI, cómo visualizarlos y dónde encontrarlos

Como ya se adelantó en el artículo del mes pasado, en el mundo de Java existe un punto intermedio entre el usuario y el programador de Java. Este lugar lo ocupa el desarrollador de páginas HTML que desea incluir la funcionalidad y vistosidad de un programa Java en sus páginas pero no tiene tiempo o ganas de aprender el lenguaje. Y es que, si bien Java es sencillo si se compara con C++, es extremadamente complejo si se compara con HTML.

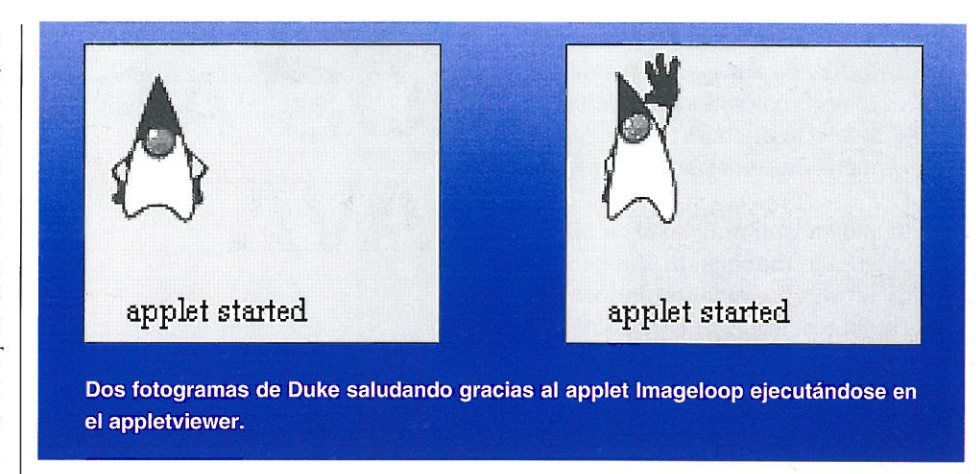
Es habitual la escena en que un usuario de Internet que desarrolla sus propias páginas o, incluso, un webmaster, quedan atraídos por la gracia de un applet Java y, cuando acuden a ver las fuentes del programa, caen desanimados al ver el esfuerzo que requiere realizar algo tan simple como una animación que, sin conocimientos de programación podrían haber realizado mediante un GIF89 (tema que se tratará en un futuro artículo). Sin embargo, el esfuerzo no sería tan grande si, tomando este ejemplo, se pudiera emplear un programa en Java ya compilado que tomara como parámetro el directorio donde se encuentran los fotogramas de una animación para proceder a mostrarla. Por ello, existen multitud de applets ya desarrollados que se pueden encontrar en BBS, Internet y servicios on-line, que pertenecen a la modalidad de freeware, por lo que pueden emplearse libremente y que cumpliendo una función general, como la de mostrar una animación, permiten su personalización mediante parámetros.

JAVA APPLETS

Los applets de Java son tratados por un browser específico de WWW como cualquier otro tipo de información como pueda ser imagen, vídeo, audio, etc. El mecanismo es análogo al de carga de una imagen. Cuando un browser detecta la etiqueta *<applet>* en el código HTML del documento que ha recuperado, carga el fichero de clases con los byte-codes correspondientes [Echeva-1]. Una vez cargado procede a su verificación, con lo que se comprueba que el código es legal y se ejecuta. El resultado de la ejecución se muestra en la página WWW como si de una imagen se tratara y, del mismo modo, el texto podrá fluir alrededor. La diferencia radica en que, ahora, la zona ocupada por el applet será la zona de E/S del programa Java. Así podrá desde mostrar un texto a interactuar con el ratón y el teclado, directamente o a través de elementos como botones, iconos, barras de desplazamiento, etc. De esta forma, una página WWW es capaz de incorporar no solamente datos sino también algoritmos para procesarlos en forma de programas ejecutables.

Para poder disfrutar de una página "Java Powered", es necesario emplear cualquier browser que muestre la marca "Java Compatible". Estos browsers definen la máquina virtual Java y proporcionan el conjunto de librerías de clases estándar necesarias para que el applet funcione. De este forma, el mismo programa podrá ser ejecutado en cualquier browser (compatible) sea cual sea el sistema sobre el que se encuentra y sin necesidad de recompilación. El resto de los browsers, como ocurría con las extensiones no estándar de HTML, simplemente ignorarán las etiquetas correspondientes y mostrarán el resto de la página.

No por encontrarse restringidos a una página de Web la funcionalidad de estos programas se verá también restringida. Así, por ejemplo un applet será capaz de gestionar comunicaciones ya sea a bajo nivel (E/S mediante sockets para redes TCP/IP), como a alto nivel (gestión de URLs). Del mismo modo, dispondrá de un conjunto de rutinas gráficas que le permitirán dibujar en pantalla a través del denominado *AWT* (*Abstract Window Toolkit*) mediante el cual se pueden crear componentes de interfaz como barras de desplazamiento, ventanas o botones. El AWT emplea el *GUI* (*Graphic User Interface*) del sistema de soporte, con lo que los elementos gráficos incorporados tendrán el mismo aspecto que cualquier otra aplicación del sistema. En definitiva, se dispondrá de todas las herramientas necesarias como complemento a un moderno lenguaje de programación.



Dos fotogramas de Duke saludando gracias al applet Imageloop ejecutándose en el appletviewer.

CONSIDERACIONES SOBRE EFICIENCIA

A la hora de realizar un applet o incorporar uno configurable en una página WWW, será conveniente tener en cuenta los siguientes aspectos sobre eficiencia:

En primer lugar, el tiempo de carga del applet. Al igual que ocurre con una imagen, el código ejecutable debe ser transmitido a través de la red. Si bien se podrá observar el resto de la página, ya que quedará reservado el lugar necesario para la ejecución del applet, si el ejecutable es muy largo, el usuario podría incluso pasar por alto la existencia del mismo, que no terminaría de llegar. Por ello, es conveniente indicar su existencia mediante, por ejemplo, el logo "Java Powered",

Una vez cargado y verificado, el applet comenzará su ejecución. Durante el arranque, a su vez, puede cargar otros recursos, por ejemplo, muestras de audio, vídeo, imágenes, más código ejecutable, lo que retrasará aún más su aparición en pantalla, por lo que nunca hay que olvidar que todos los recursos se van a transmitir a través de Internet.

Sin embargo, una vez cargados todos los recursos se ejecutará en la máquina local, por lo que pasará a ser un programa más en el escritorio de trabajo, con lo que presentará una total interactividad. Así por ejemplo, si se deseara mostrar una molécula en 3D en movimiento mediante CGI, sería necesario enviar a través de la red cada fotograma, mientras que utilizando Java, una vez cargado el applet ejecutable y los datos que definen la molécula, el proceso de representación, y renderizado se realiza totalmente en local y de forma interactiva. La diferencia de velocidad es tal que en el primer caso el empleo de este tipo de páginas es del todo imposible y, en el segundo, una realidad.

Otra de las ventajas de los applets Java, es que pueden quedar atrapados en las cachés de servidores proxy [Echeva-2] y en la misma caché del browser que los visualiza pudiendo, incluso, ser salvados a disco.

INCLUSIÓN DE UN APPLET EN UNA PÁGINA WWW

Como ya se ha indicado, para incluir un applet Java en una página de WWW se emplea la etiqueta *<applet>* de HTML. Para facilitar parámetros a un applet, que al fin y al cabo es un programa ejecutable, se define otra nueva etiqueta: *<param>*.

Estas etiqueta no son estándares, por lo que serán ignoradas por aquellos browsers que no sean "Java Compatible". Este hecho es inteligentemente aprovechado a la hora de definir un contenido alternativo a ser mostrado en este último caso. Así, la etiqueta es doble:

<applet atributos>
parámetros
contenido alternativo
</applet>

donde parámetros es una sucesión indefinida dependiente del applet concreto de etiquetas *<param>*.

De esta forma todo el código HTML que se encuentre envuelto por la misma, a excepción de los parámetros del applet, será ignorado por un browser "Java Compatible". Otro browser, en cambio, lo que ignorará serán precisamente las nuevas etiquetas, con lo que la compatibilidad queda garantizada. Así por ejemplo, la secuencia:

<applet code=ImageLoopItem.class width=80 heigth=90>
<param name=nimgs value=10>
<param name=img value=duke>
<param name=pause value=1000>
<img src=duke.gif>
</applet>

mostrará una imagen de "Duke", la mascota de Java sea cual sea el browser con que se acceda a la página, aunque, en uno "Java Compatible", la imagen se verá en movimiento. También suele ser costumbre indicar la existencia de applets a los que no sea tan sencillo encontrar un sustituto, de la forma:

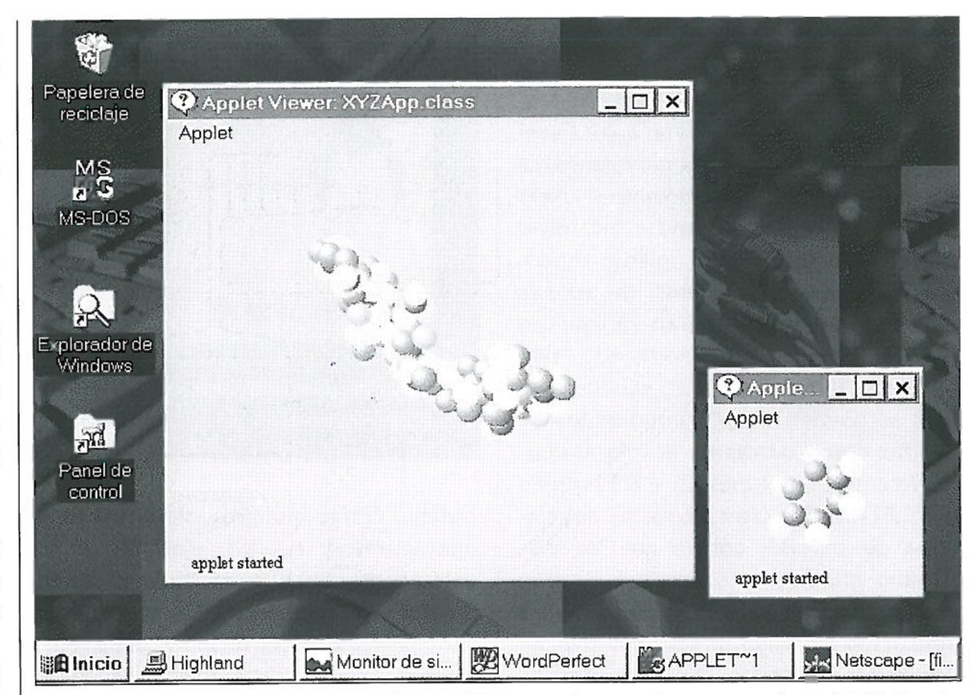
<applet code=AppletAtomico.class with=100 heigth=100>
No vuelvas por aquí hasta que no te busques un
browser <I>Java Compatible</I>
</applet>

Como se puede apreciar en los ejemplos, la etiqueta *<applet>* va acompañada de ciertos atributos, algunos de los cuales son obligatorios mientras que otros son opcionales. Todos los atributos, siguiendo la sintaxis de HTML se especifican de la forma: *atributo=valor*. Los atributos obligatorios son:

code: Indica el fichero de clase ejecutable, que suele tener una extensión

.class. No se permite un URL absoluto en este atributo, aunque sí puede ser relativo al atributo opcional *codebase*.
width: Indica la anchura inicial que el browser debe reservar para el applet en pixels.
heigth: Indica la altura inicial en pixels. Un applet que disponga de una geometría fija no se verá redimensionado por estos atributos. Por ello, si los atributos definen una zona menor que la que el applet utiliza, únicamente se verá parte del mismo, como si se visualizara a través de una ventana, eso sí, sin ningún tipo de scroll.
Los atributos opcionales comprenden:
codebase: Se emplea para utilizar el URL base del applet. En caso de no especificarse, se empleará el mismo que tiene el documento.
alt: Funciona exactamente igual que el atributo *ALT* (recuérdese que HTML no es *case sensitive* -sensible a mayúsculas/minúsculas-) de la etiqueta *<img>*, es decir, muestra un texto alternativo, en este caso al applet, en browsers en modo texto o que entiendan la etiqueta *<applet>* pero no implementen una máquina virtual Java..
name: Otorga un nombre simbólico a esta instancia del applet en la página que puede ser empleado por otros applets de la misma página para localizarlo. Así, un applet puede ser cargado varias veces en la misma página tomando un nombre simbólico distinto en cada momento.
align: Se emplea para alinear el applet permitiendo el texto fluir a su alrededor. Funciona igual que el atributo homónimo de la etiqueta *<img>* según las extensiones de Netscape descritas en [Echeva-3], pudiendo tomar los valores *left, rigth, top, texttop, middle, absmiddle, baseline, bottom y absbottom.*
vspace: Indica el espaciado vertical entre el applet y el texto. Se emplea de la misma forma que con el atributo *<img>*.
hspace: Funciona de manera análoga a *vspace* pero indicando espaciamiento horizontal.

Es probable encontrar en algunas distribuciones (sobre todo, de HotJava) otras etiquetas para la inclusión de applets, como *<APP>*. Esto se debe a que ésta es la tercera revisión de la extensión de HTML para incrustación de applets y ha sido adoptada como la definitiva. Por ello, cualquier otro medio corresponde a implementaciones obsoletas que han quedado descartadas.



El appletviewer aplicado a una página HTML que invoca dos veces al applet Molecule Viewer (las moléculas siguen los movimientos del ratón).

Sintaxis de las etiquetas <applet> y <param>
<applet code= width= heigth=
[codebase=][alt=][name=]
[align=][vspace=][hspace=]>
<param name= value=>

Atributos obligatorios
code: Nombre de la clase principal
width: Anchura inicial
heigth: Altura inicial

Atributos opcionales
codebase: URL base del applet
alt: Texto alternativo
name: Nombre de la instancia
align: Justificación del applet
vspace: Espaciado vertical
hspace: Espaciado horizontal

ALGUNOS APPLETS ÚTILES

Con la revista se proporcionan algunos de los applets de propósito más general que rondan por Internet con el fin de que el lector los pueda incorporar a sus propias páginas. En cualquier caso, el lugar por excelencia para buscar applets es ***Gamelan: http:// www.gamelan.com.*** Basta visitar este

LISTADO 1

```
<html>
<head>
<title>Ejemplo de applet incrustado</title>
</head>

<body>
<hr>

<H1>Ejemplo de uso de un applet incrustado</H1>

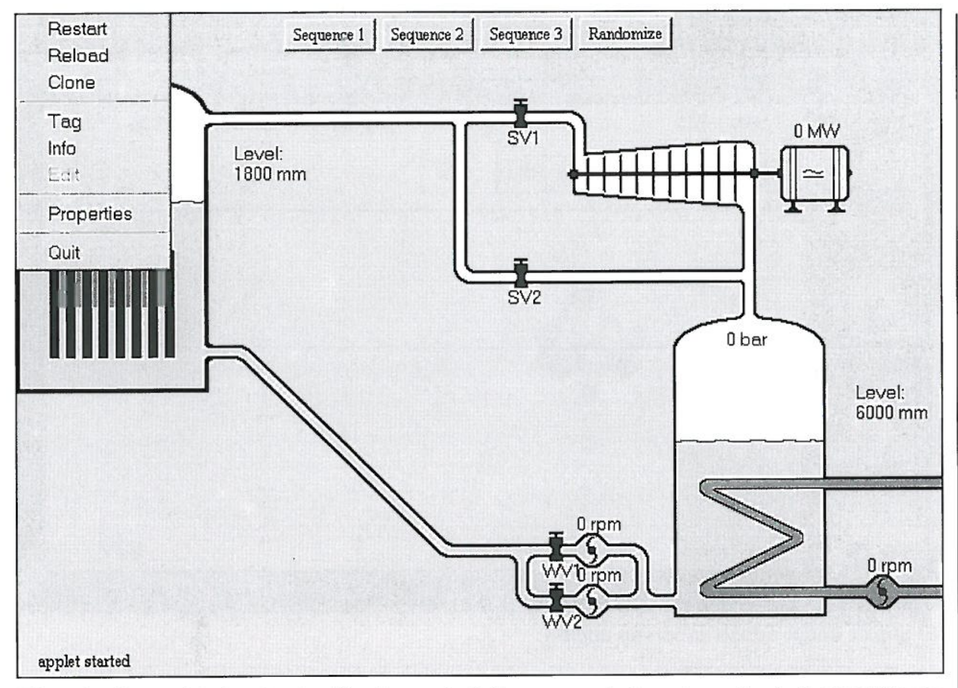
<applet code="Chart.class" width=300 height=75>
<param name=title value="Semana corriente">
<param name=columns value="2">
<paran name=scale value="10">
<param name=orientatiön value="horizontal">
<param name=c1 value="5">
<param name=c1_style value="solid">
<param name=c1_color value="red">
<param name=c1_label value="Dias Laborables">
<param name=c2 value="2">
<param name=c2_style value="striped">
<param name=c2_color value="green">
<param name=c2_label value="Dias Festivos">

<B>Semana corriente</B><br>
<I>Dias laborables</I>: 5<br>
<I>Dias festivos</I>: 2

</applet>

<hr>
</body>

</html>
```



Ejemplo de applet de simulación de control de una central nuclear. En la ilustración se pueden observar las opciones del menú del appletviewer.

servidor para convertir las propias páginas en espectáculos de imagen y sonido interactivos mediante la utilización de los miles de applets que allí quedan registrados.

ImageLoop

Uno de los applets (desarrollado por James Gosling) más útiles y sencillos para incorporar animaciones a las páginas WWW. Consta de los siguientes parámetros:

img: URL en el que figuran las imágenes que forman la animación. Éstas deben ser GIFs y nombrarse T1.gif, T2.gif, etc.

nimgs: número de imágenes de que consta la animación.

pause: pausa en milisegundos entre bucles de la animación.

Así, un ejemplo de uso sería:

<applet code=ImageLoopItem width=80 heigth=90>
<param name=nimgs value=10>
<param name=img value=duke>
< param name=pause value=1000>
</applet>

que buscaría los ficheros T1.gif ... T10.gif del directorio duke para generar la animación.

AudioItem

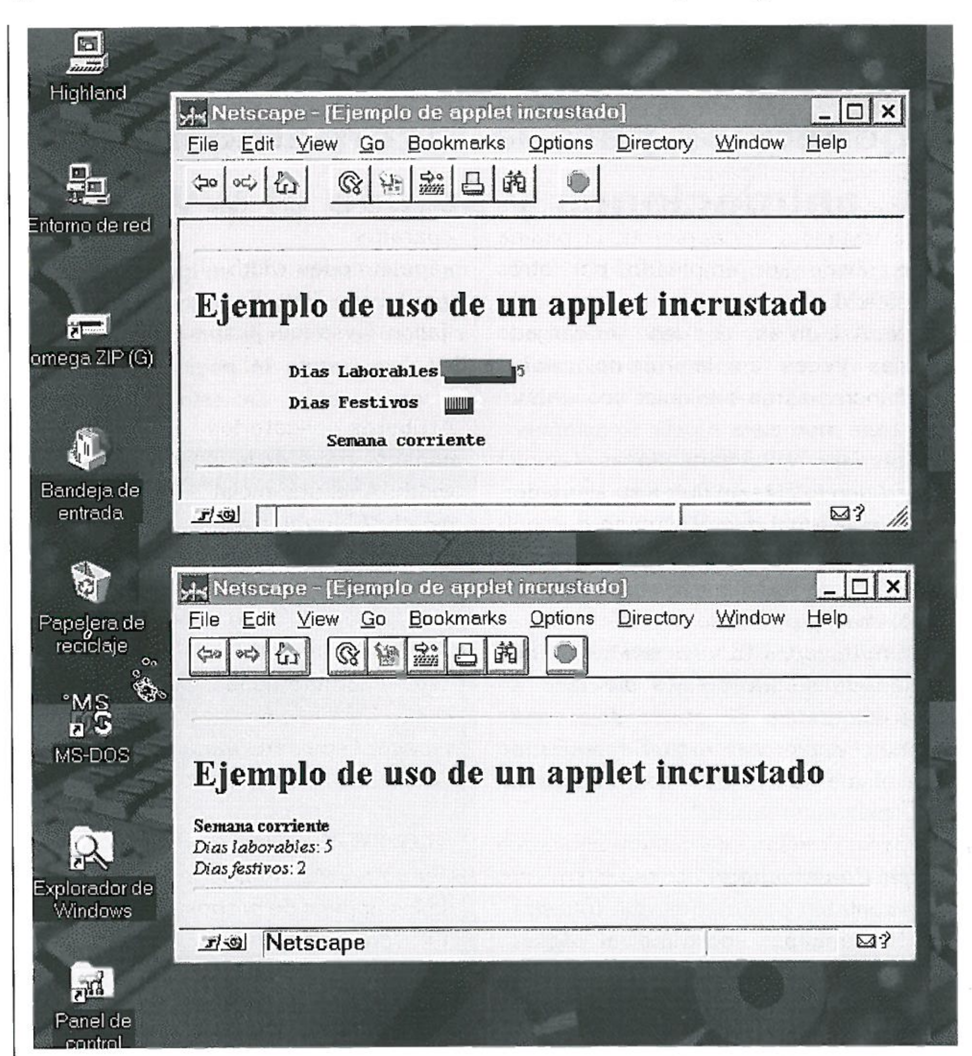
Interpreta un fichero de sonido de entre una secuencia que se le facilita como parámetro. Cada vez que el usuario pulsa sobre el icono de sonido que muestra, se pasa al siguiente fichero. Toma como único parámetro *snd* para indicar los URLs de los ficheros de sonido a reproducir, que deben ir separados por el símbolo "|". Por ejemplo:

<applet code="AudioItem.class" width=15 height=15>
<param name=snd value="Hola.au|fichero2.au|adios.au">
</applet>

Animator

Animator es un applet de propósito más general que ImageLoop ya que permite generar animaciones con audio y desplazamiento. Para ello, emplea los siguientes parámetros, aunque no todos son obligatorios:

imagesource: URL del directorio que contiene las imágenes. Éstas deben nombrarse T1.gif, T2.gif, etc.

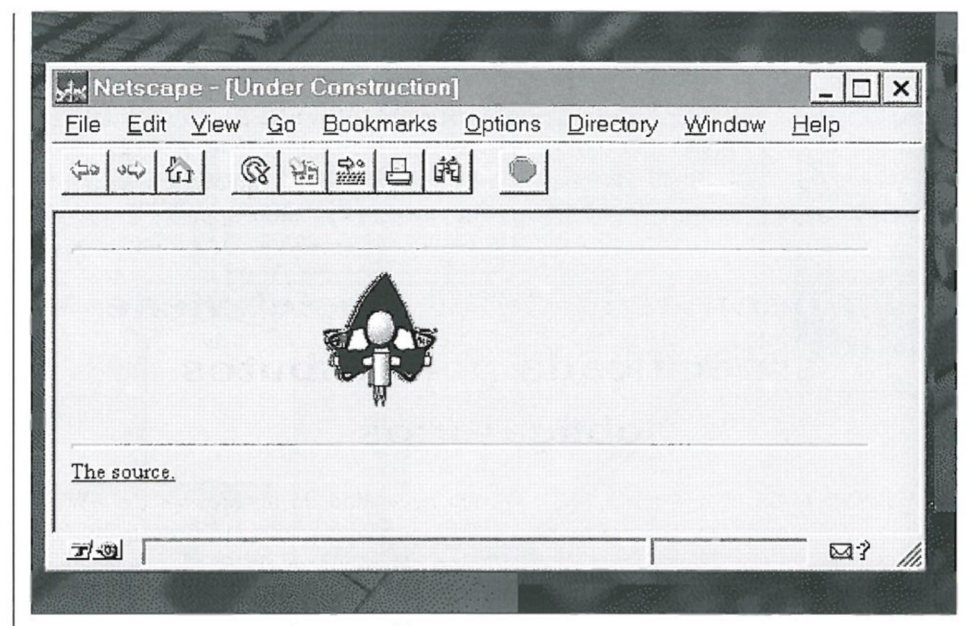


Aspecto de la página HTML correspondiente al Listado 1 visualizada mediante un browser "Java Compatible" (arriba) y con otro browser que no incorpora Java.

startup: URL de una imagen que se presentará mientras se carga el resto a través de la red.
background: URL de una imagen de fondo.
startimage: Índice del primer fotograma.
endimage: Índice del último fotograma.
pause: Pausa entre fotogramas en milisegundos.
pauses: Lista de pausas entre fotogramas en milisegundos separadas por el carácter "|". Tiene preferencia sobre el parámetro anterior. Por ejemplo: 100|0|400|1|...
repeat: true o false, indica si la animación se repite al terminar
positions: Lista de posiciones de cada fotograma en pantalla. Cada posición se indica según coordenada_x@coordenada_y y se separa de las demás mediante el carácter "|". Así, se pueden indicar desplazamientos por pantalla. Por ejemplo: 100@10|100@15|100|20.
images: Índiices de las imágenes. Permite ejecutar la animación colocando los fotogramas en cualquier secuencia. También sirve para repetir fotogramas. Por ejemplo: "1|2|3|2|3|2|3|4"
soundsource: URL del directorio que contiene los ficheros de audio.
soundtrack: URL relativo a *soundsource* del fichero de audio que se repetira como fondo de la animación.
sounds: Lista de URLs de cada fichero de audio que se interpretará asociado a cada fotograma. Se puede dejar una entrada vacía para aquel fotograma para el que se desea silencio. Por ejemplo: "hola.au||adios.au"

Se han desarrollado applets genéricos para la incorporación de animaciones y sonido al WWW

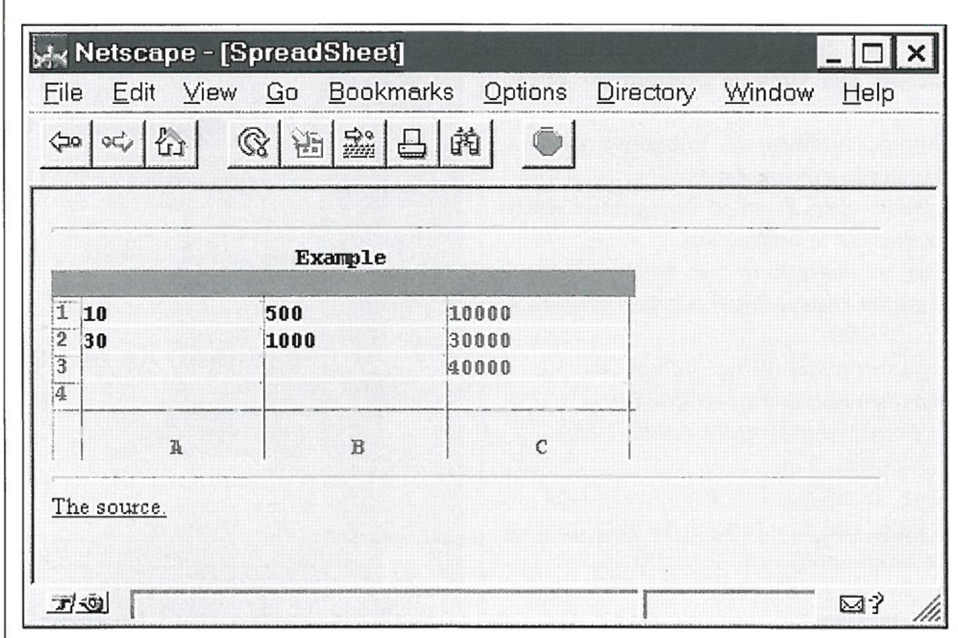
Under Construction
Un applet sin parámetros que muestra a Duke taladrando para indicar página en construcción. Dispone de sonido. Como ejemplo:
<applet code="JackhammerDuke.class" width=300 heigth=100>
</applet>



El applet under construction en acción.

Bar Chart
El Bar Chart es uno de los applets realmente útiles, especialmente si se combina con un script CGI, ya que si no, cumple simplemente las funciones de una imagen. Lo idóneo es la realización de un CGI que genere la página HTML que invoca al applet. De esta forma, se le pueden pasar parámetros variables en función de la información que se desea mostrar. En una palabra: el CGI generaría resultados y el applet los mostraría. Este es un ejemplo que viene a defender la teoría de que CGI y Java son complementarios y no antagónicos y pueden convivir en armonía. Así, se propone como ejercicio al lector que modifique la calculadora básica CGI de [Echeva-4] para mostrar los resultados en forma de gráfico de barras mediante la utilización de este applet.

La configuración del applet se realiza mediante los parámetros:



Mediante Java es posible realizar hojas de cálculo que se ejecuten localmente en una página de WWW.



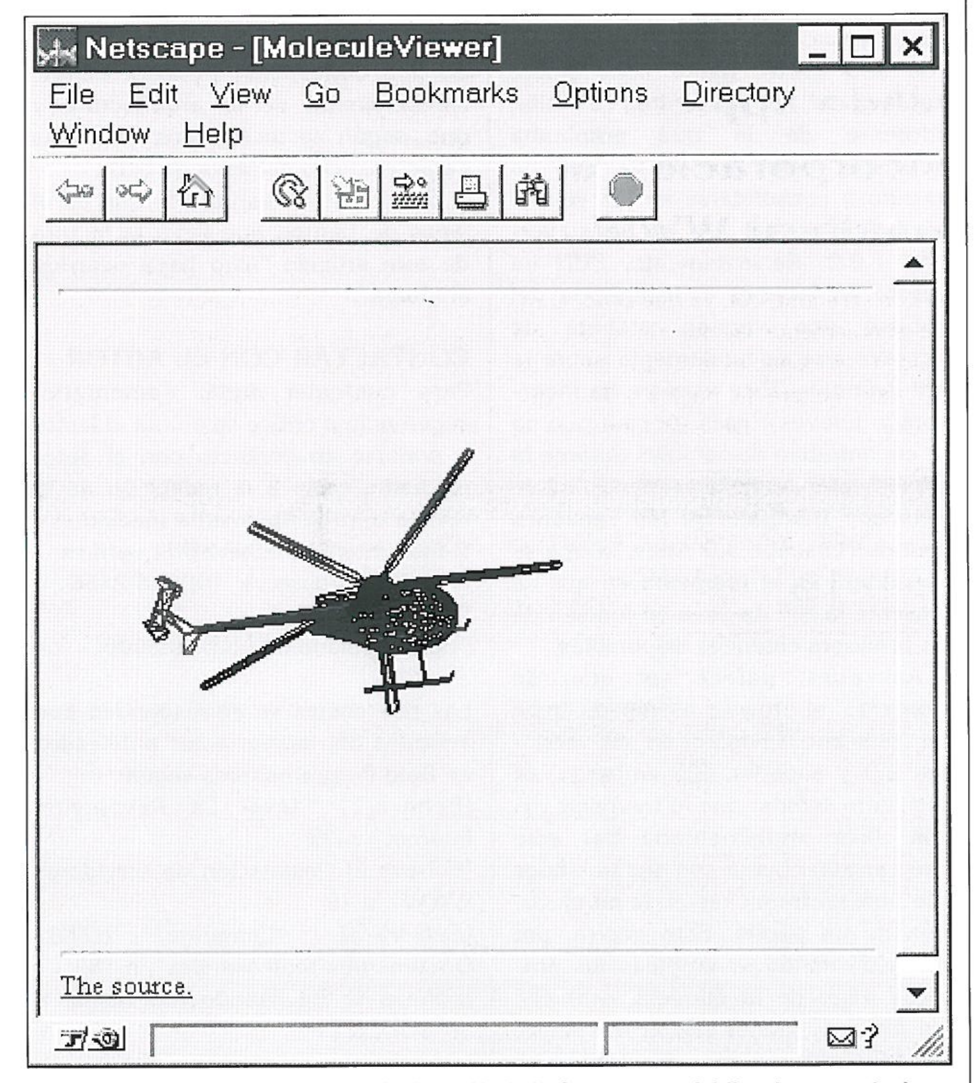
title: Título del gráfico
columns: Número de conceptos
orientation: Orientación del gráfico, puede ser *horizontal* o *vertical*.
scale: Pixels por unidad en el gráfico.
cX_style: Apariencia de la barra número X: *solid* o *stripped* (entramada).
cX: Valor de la barra número X.
cX_label: Etiqueta para la barra número X.
cX_color: Color de la barra número X. Puede ser *green*, *blue*, *pink*, *orange*, *magenta*, *cyan*, *white*, *yellow*, *gray* o *darkGray*.

Así, el ejemplo de la figura 1. Se ha realizado mediante el código del listado 1:

APPLETVIEWER: EL VISOR DE APPLETS DEL JDK

El JDK de SUN incluye un visor de applets denominado *appletviewer*. Este programa permite ejecutar applets Java sin necesidad de un browser de WWW. Para ello, es capaz de realizar una conexión para acceder al URL que se le especifique y mostrar cada applet indicado en ese documento en su ventana. Para ejecutarlo, se le pasa un fichero HTML o un URL como parámetro y el programa se encarga de leerlo y mostrar los applets que incluya. En caso de que el URL o página HTML tenga invoque varios applets o que se le pase más de un argumento, el appletviewer mostrará varias ventanas, una para cada applet.

La geometría de un applet viene especificada por atributos obligatorios



Otro ejemplo de las capacidades de Java. Un helicóptero vectorial lineal que evoluciona siguiendo los movimientos del ratón.

BIBLIOGRAFÍA

- SUN Microsystems' Java Home Page
http://www.hotjava.com
- Gamelan
http://www.gamelan.com
- News
news:comp.lang.java
- Arthur van Hoff, Shami Shiao, Orca Starbuck, SUN Microsystems Inc. "Hooked on Java", Addison-Wesley,1996.

Se ejecuta según la sintaxis:
appletviewer [opciones] URLs ...
y admite la opción *-debug* para comenzar una sesión con el depurador *jdb*.

Todas las ventanas del appletviewer cuentan con un menú con una opción Applet que despliega un submenú con las siguientes opciones:
Restart: Para la ejecución del applet y lo rearranca. Esto no funciona siempre ya que para ello, es preciso que el applet contemple los métodos start y stop, de los que se hablará en los capítulos dedicados al lenguaje.
Reload: Carga nuevamente el applet. Es de utilidad si se ha recompilado el mismo, para no tener que salir y rearrancar el appletviewer.
Clone: Crea una copia del applet ejecutándose en otra ventana. El nuevo applet comienza a ejecutarse desde el principio.
Tag: Muestra una ventana con la etiqueta HTML que invoca al applet con la geometría de la ventana en ese momento. Es interesante para cortar y pegar en una página HTML.
Info: Muestra información adicional, si existe, sobre el applet en una ventana.
Properties: Muestra una ventana de configuración de control de seguridad y red.
Close: Cierra la ventana. El programa no termina hasta que se han cerrado todas las ventanas que ha generado.

PROPIEDADES DEL APPLETVIEWER

A través de la ventana de propiedades del appletviewer es posible definir los servidores proxy de HTTP y firewall [Echeva-2] y sus puertos, el control de

seguridad del acceso a red (*Network Access*) y el tipo de acceso a clases (*Class Access*).

El Network Access puede ser:

None: No permite a los applets acceso a la red.

Applet Host: Los applets sólo pueden acceder a información que se encuentre en el mismo servidor del que fueron cargados. Éste es el modo por defecto y el único que acepta Netscape 2.x. Fue un fallo en la implementación de este modo lo que provoco el "agujero del Netscape" [Echeva-1]

Firewall: Los applets que provienen de máquinas externas al firewall sólo pueden acceder a recursos externos al mismo y los applets internos pueden acceder a cualquiera.

Unrestricted: Cualquier applet puede realizar conexiones con cualquier máquina de Internet. No se recomienda el uso de este medio salvo en casos muy contados, por ejemplo, el desarrollo de applets cooperativos que puedan situarse en cualquier lugar de Internet para realizar, por ejemplo, *multichat*.

El *Class Access* controla las classes a las que un applet puede acceder y se le pueden dar los valores:

Restricted: Exige que una clase sea estándar y se encuentre presente en todos los browsers "Java Compatible" para permitir el acceso a la misma.

Unrestricted: Es el modo por defecto y permite que los applets accedan a cualquier clase que deseen, aunque no se encuentre disponible en todos los browsers "Java Compatible"

HOTJAVA VS NETSCAPE

Según acuerdos comerciales, SUN Microsystems y el resto de empresas que licencien la tecnología Java, conservan separados sus segmentos de mercado ya que, en caso contrario, SUN podría convertirse en competencia directa de aquellos a los que proporciona la tecnología, lo que se podría calificar de competencia desleal. Esto podría dar lugar a tensiones como ha ocurrido en otras ocasiones entre famosas empresas desarrolladoras de software y sistemas operativos o entre otras proveedoras de infraestructura de telecomunicaciones y sus clientes, casos que no procede abordar en este artículo. Por ello, SUN se ha apresurado a informar de que su browser HotJava es, simplemente, un sistema de demostración, ya que no consideran su actividad el desarrollo de browsers, lo que reservan para terceros como Netscape Corporation. Por otra parte, hasta hace poco, HotJava se acogía a una API alfa, diferente de la que empleaba Netscape, beta, que es la que ha pasado a consolidarse en las versiones definitivas del JDK, ya por su versión 1.0.2. Recientemente, SUN ha puesto en Internet a disposición del público una version prebeta de HotJava que se fundamenta sobre la API definitiva. Esta versión, ha incorporado nuevas clases (pocas) que no se encontraban disponibles durante la implementación de la versión 2.0.2 de Netscape Navigator ni las versiones beta de Netscape 3.0 Atlas, lo que se concretará en el momento en que se presente la API de Java en función de los browsers entonces disponibles.

Un buen desarrollador de páginas WWW debe saber buscar la solución más adecuada, que no siempre es la más vistosa

Así pues, parece ser que, de momento, el browser a emplear debe ser Netscape Navigator en sus versiones 2.0 y superior. Sin embargo, es necesario señalar que la máquina virtual Java implementada por este browser aún cuenta con algunos bugs que pueden traducirse en la no ejecución de un applet. Esto ocurre, por ejemplo cuando se emplean los atributos *align* de la etiqueta *<applet>*. Así pues, al menos por el momento, conviene no emplear este atributo en las páginas WWW y, si en alguno de los ejemplos no aparece un applet que debería aparecer, comprobar si se ha empleado para quitarlo.

CONCLUSIÓN

Hay que recordar que Java es mucho más que WWW y que se basa en una arquitectura *network centric* [Echeva-1] que asume un umbral mínimo de calidad de red. Si bien en este momento no se alcanza ese umbral, el mes pasado ya se comentaron las líneas de I+D (investigación y desarrollo) que se estaban siguiendo para lograr este objetivo a corto-medio plazo.

Sistemas como Java parecen ser claves en el futuro. Tanto es así que empresas de la talla de Lucent Technologies, que quizá el lector conozca mejor por su antiguo nombre, AT&T Network Systems, se han apuntado a la carrera desde la competencia mediante el desarrollo del lenguaje *Inferno*, otro lenguaje basado en una concepción *network centric* independiente de la arquitectura y que, según se dice, reúne todas las características del Java de SUN.

En definitiva, es posible que en el lapso de tiempo que lleva la lectura de este artículo "algo haya ocurrido ahí fuera".

CONTACTAR CON EL AUTOR

Para cualquier duda, comentario, sugerencia o crítica se anima al lector a ponerse en contacto con el autor mediante carta a la redacción de la revista o, preferiblemente mediante:

E-mail Internet: echeva@dit.upm.es
E-mail Compuserve: 100646,2456
WWW: http://highland.dit.upm.es:8000

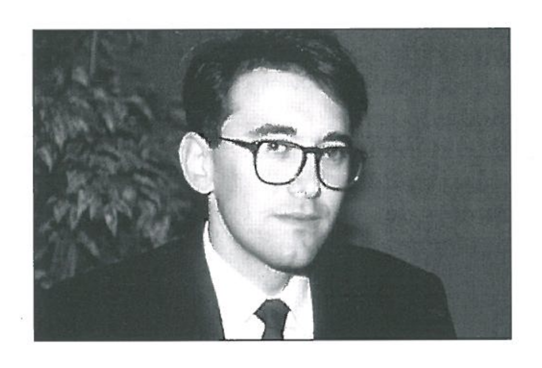
Las referencias se corresponden con artículos del mismo autor publicados en Sólo Programadores según:

[Echeva-1] "Java: La Revolución Internet", n.22
[Echeva-2] "Instalación de servidores WWW", n.19
[Echeva-3] "Lenguaje HTML: Documentos profesionales", n.18
[Echeva-4] "Facilitando Información a un CGI", n.21

VISUAL BASIC ENTRA EN LAS AUTOPISTAS DE LA INFORMACIÓN

Desde que en 1973, Internet fuera creada por la Agencia de Proyectos de Investigación de Defensa de los EE.UU. (DARPA), con el fin de asegurar que sus sistemas de comunicación continuasen funcionando en caso de guerra, y de sus posteriores usos, principalmente en materia académica y de investigación, la Red ha evolucionado mucho. Ya no se trata de un instrumento elitista como lo fue años atrás, sino que cualquier persona o compañía puede utilizar sus recursos, a la vez que son ya millones de usuarios los que han reconocido su enorme potencial.

Por esta razón, cada vez son más los programadores que están desarrollando aplicaciones para Internet. Con el fin de satisfacer la demanda de estos millones de usuarios, Microsoft ha querido facilitarles el trabajo, de tal forma, que no requieran meses de aprendizaje en el uso de una nueva herramienta de desarrollo y permitir así al desarrollador aprovechar la ventaja competitiva, que significa utilizar un lenguaje que ya conoce para desarrollar aplicaciones para Internet.



Del mismo modo, que Visual Basic facilita el desarrollo de aplicaciones basadas en Windows, Visual Basic Script, un lenguaje de escritura (scripting) para Internet basado en Visual Basic, facilita el desarrollo de aplicaciones para la Red.

Visual Basic Script: un lenguaje abierto

Visual Basic Script ofrece capacidades de escritura, automatización y desarrollo a la medida, para los visualizadores World Wide Web (WWW), al mismo tiempo que se define como un subconjunto del lenguaje Visual Basic, totalmente compatible con él. En este sentido, Visual Basic Script permite a los desarrolladores escribir código Visual Basic, que reside en los documentos HTML (Language Markup HyperText). Cuando un visualizador alcanza el "·ScriptÒ tag", llama a Visual Basic Script para compilar el código. En este caso, el código es ligado al evento "click" sobre un botón denominado "Comando", y la salida es escrita en una caja de texto llamada "MyHTMLListBox". Asimismo, hay que tener en cuenta que el código de Visual Basic Script es representado como el texto ASCII en el documento HTML, y es compilado sobre el "download". En este sentido, el visualizador necesita tener soporte para Visual Basic Script y debe poder integrar facilidad de "scripting" con controles o "applets" vinculados en la corriente HTML. Esto es lo que las futuras versiones de Microsoft Internet Explorer y diferentes visualizadores de otras compañías, que licencien la familia de herramientas para Internet de Microsoft, llevarán a cabo.

Asimismo, señalar que Visual Basic Script es muy fácil de obtener, ya que el programador puede verlo soportado por diferentes visualizadores sobre múltiples plataformas, al mismo tiempo que puede obtenerse en Internet sin ningún coste.

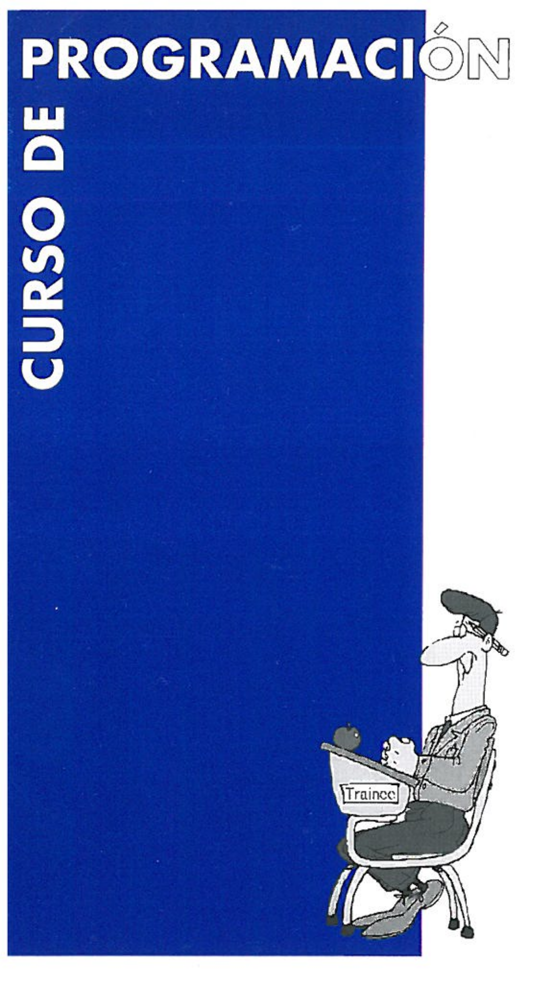
La Automatización OLE es otro beneficio clave de Visual Basic Script, ya que puede ser utilizado para manipular tanto el visualizador (futuras versiones de Microsoft Internet Explorer incorporarán un interface de Automatización OLE) como otras aplicaciones sobre el sistema de sobremesa. Una de sus principales características es que puede ser utilizado para asignar propiedades y métodos sobre los controles OLE (ficheros .OCX) y "applets" creados por el lenguaje Java de Sun Microsystems, que son contenidos dentro de una página HTML.

Todo esto nos ofrece una perspectiva técnica de lo que supone Visual Basic Script, pero debemos plantearnos, asimismo, qué significa o qué ofrece para los usuarios o visitantes de un Web. Hoy en día la mayoría de las páginas Web en Internet consisten simplemente en una representación gráfica o de texto estática. En este sentido, Visual Basic Script ofrece actividad e interés a las páginas Web. El programador puede crear páginas que respondan preguntas, consultas, o cuestiones, o bien que comprueben los datos del usuario, calculen expresiones, realicen enlaces con otras aplicaciones y conecten con controles OLE y animaciones en 3 D. En definitiva, Visual Basic Script permite que los programadores de Visual Basic hagan interesantes y atractivas las "páginas muertas" de Internet para los usuarios.

José Antonio Alvarez
Jefe de Producto de Herramientas de Desarrollo de Microsoft Ibérica

BÚSQUEDA RECURSIVA DE ARCHIVOS

José C. Remiro



Seguimos dentro de esta sección estudiando ejemplos prácticos para aplicar los conocimientos expuestos en los capítulos anteriores. Combinando la recursividad con los servicios ofrecidos por el DOS puede implementarse fácilmente un algoritmo para la búsqueda de archivos

Una de las principales funciones de un sistema operativo es ocultar la complejidad del hardware al usuario, de forma que su utilización esté más cercana al lenguaje humano que al lenguaje máquina. Así, desde el punto de vista físico, un archivo puede verse como un conjunto de perturbaciones magnéticas en la superficie del disco, sin embargo, desde el punto de vista lógico un archivo es una secuencia de sectores dentro del disco (comprendiendo un sector un conjunto fijo de bytes adyacentes) teniendo este conjunto de sectores una unidad a nivel de sistema operativo. Éste, no es más que un intermediario entre ambas formas de entender los archivos.

La organización de la información dentro de un disco DOS, es la siguiente: a cada dispositivo de almacenamiento secundario se le asigna una unidad, representada por una letra. Cada unidad presenta un único directorio raíz, a partir de él se insertan nuevos archivos o directorios, siendo un directorio un archivo especial que guarda información de los archivos y directorios que pudieran estar contenidos en él. Esto permite por un lado estructurar de forma jerárquica la información almacenada en un dispositivo de almacenamiento secundario, restringiendo la búsqueda de archivos a lo largo del dispositivo y por otro, mantener el espacio de almacenamiento no utilizado en forma de directorio libre de archivos.

Para utilizar la estructura de archivos del DOS, bien se pueden utilizar las funciones que el sistema operativo proporciona o bien se pueden saltar estas funciones e ir directamente al lugar donde el sistema operativo almacena la información para su uso particular, es decir, utilizar la FAT (Tabla de asignación de archivos). Si bien es cierto que para la construcción de utilidades de bajo nivel se necesita acceder a la FAT, existe información útil que proporciona el sistema operativo para realizar búsquedas repetidas en un directorio y que es creada y borrada tras completarse la búsqueda, además esta información no es necesario que se grabe en el disco, estando almacenada en el área de transferencia de disco (DTA).

En este artículo se presentará información acerca de como manipular la estructura de archivos y directorios a través del DTA.

FUNCIONES DEL DOS

Hay dos llamadas de función del DOS que acceden a un directorio y buscan las entradas que concuerdan con un especificador de archivo. Se entenderá por especificador de archivo a una trayectoria absoluta o relativa de archivo o directorio, pudiendo incluir cualquiera de los caracteres comodín (* y ?).

La interrupción 21H, permite realizar llamadas a las funciones del DOS, de entre éstas interesan las que se corresponden con leer e inspeccionar los directorios de un disco. La forma de ejecutar las interrupciones desde un lenguaje de alto nivel es casi siempre la misma: se introducen los diferentes valores de entrada en los registros del microprocesador (número de servi-

cio, número de interrupción y los posibles parámetros del servicio) y tras ser ejecutada la interrupción, se devuelve la salida en los registros. Normalmente los registros del micro están representados por una estructura especial dentro del lenguaje. Además, es posible que el lenguaje enmascare estas llamadas mediante funciones que utilizan directamente las interrupciones del DOS (como se verá más adelante las funciones *FindFirst* y *FindNext* pertenecen al anterior conjunto).

ESTRUCTURA DEL DTA Y FUNCIONES ASOCIADAS

El área de transferencia de datos contiene la siguiente información:

• Una cabecera de 21 bytes, utilizadas en exclusiva por el DOS para grabar información sobre las llamadas. La modificación de alguno de estos 21 bytes impedirán la correcta búsqueda de archivos.
• Un byte que almacena información acerca de los atributos asociados al archivo.
• La fecha y hora de la última modificación del archivo en formato compacto.
• El tamaño en bytes del archivo.
• El nombre y la extensión del archivo.

Toda esta información que proporciona el DOS, es utilizada en Pascal a través de la estructura de datos *SearchRec* que se describe en el cuadro 1.

Como se ha trabajado hasta ahora con archivos parece ser que la forma adecuada de trabajar éstos, es abrir un archivo y determinar sus propiedades, este procedimiento es el adecuado cuando se trabaja con un archivo del que se conoce su nombre completo, pero no permite especificar un conjunto de archivos a través de los caracteres comodín.

DOS, permite trabajar con una especificación de archivos a través de las funciones $4E y $4F, que funcionan de la siguiente forma: a través de la función $4E se pasa una especificación de archivo completa (trayectoria absoluta), esta función devuelve la primera ocurrencia que concuerde con el parámetro que se le ha pasado. Si encuentra al menos uno, inicializa el DTA de tal forma que a través de la función $4F se pueda localizar las siguientes entradas para la misma especificación de archivo. Si no se encontrase ninguna ocurrencia, entonces en el registro AX se almacenaría un código de error apropiado.

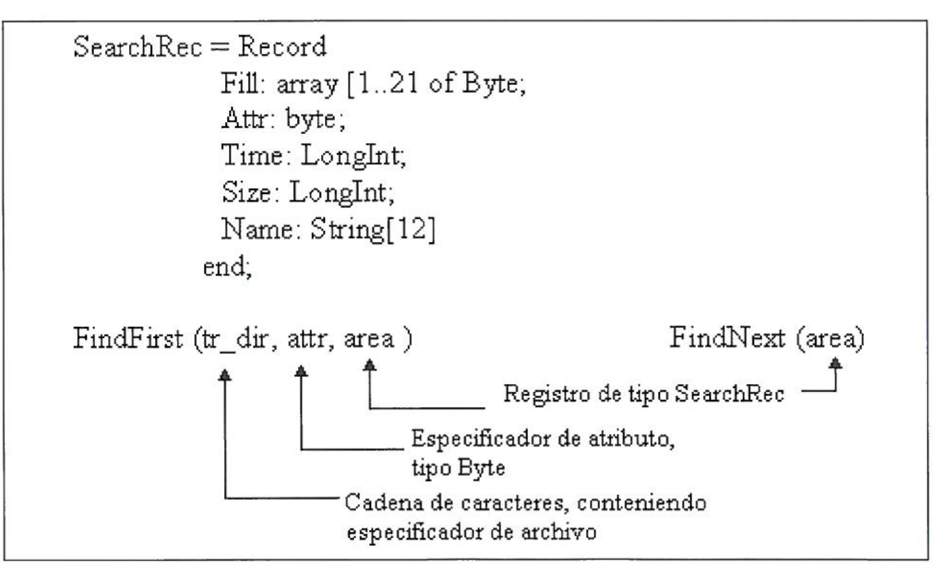
La mayoría de los lenguajes de alto nivel disponen de estas llamadas a las funciones del DOS pero simplificando el trabajo al programador. En el caso de Turbo Pascal, las llamadas a las funciones $4E y $4F se corresponden con los procedimientos *FindFirst* y *Findnext*, respectivamente. *FindFirst* admite como parámetros una cadena de caracteres, que contendrá un especificador válido de archivo, una variable de tipo *word* que especificará el conjunto de atributos que deberá tener el archivo y por último una estructura de tipo *SearchRec*, que contendrá el DTA pero con un formato más manejable por el programador. Tras la ejecución *FindFirst* no será necesario comprobar el valor devuelto en el registro AX, pues la función *DosError* también facilita la comprobación del éxito de la búsqueda. Si *FindFirst* ha encontrado al menos un archivo que concuerde con la especificación y los atributos, la función devolverá el valor 0, en caso contrario devolverá otro valor (por ejemplo, 2 archivo no encontrado, 18 no hay más archivos que localizar etc.). En cuanto al procedimiento *FindNext*, sólo admite como parámetro una variable de tipo *SearchRec*, que permite seguir buscando, eso sí, con la especificación de archivo del último *FindFirst* ejecutado con éxito.

DESCRIPCIÓN DEL PROGRAMA

El programa que acompaña al artículo trata de mostrar cómo se pueden utilizar los servicios del DOS en un lenguaje de alto nivel, y cómo puede simplificarse un algoritmo utilizando la recursividad cuando esta se aplica en situaciones adecuadas.

Al DOS, le acompañan una serie de utilidades, pero desgraciadamente no dispone de ninguna herramienta para realizar búsquedas de un archivo o conjunto de archivos a lo largo de la estructura de directorios. El programa BUSCAR.PAS (en el cuadro 2 se muestra el cuerpo del programa principal) pretende ser un comando para la localización de archivos.

En cuanto a la interfaz con el usuario, será de lo más simple. El usuario junto con el nombre del programa introducirá cero o un parámetro, si se introduce alguno más se ignorará. Si no se introduce parámetro alguno se producirá la búsqueda de cualquier archivo en



Cuadro 1.

```
begin {cuerpo del programa}
 ocultar_cursor (aux_cx);
 clrscr;
 trayectoria:= expandir;
 dividir_trayectoria (trayectoria, directorio, archivo);
 if unidad_preparada (trayectoria)
   then
    begin
     localiza_archivos(directorio, archivo);
     clrscr;
     end
   else
    writeln ('Unidad no preparada');
 visualizar_cursor (aux_cx);
end.
```

Cuadro 2.

el directorio actual de trabajo de la unidad activa y además de forma recursiva, es decir, se mostrará el contenido de sus subdirectorios (si los hubiera). Por el contrario, si el usuario introduce un parámetro, éste deberá ser una trayectoria absoluta o relativa que podrá incluir caracteres comodín. El programa devolverá todos los nombres de archivos que se correspondan con el especificado por el parámetro, junto con la información disponible de dichos archivos, pero incluyendo aquellos que se encuentren dentro de los subdirectorios del directorio introducido como parámetro.

Para eliminar el cursor de la pantalla mientras se ejecuta el programa y posteriormente devolver el aspecto que tenía el cursor, se realiza la llamada al procedimiento *ocultar_cursor* que admite como parámetro una variable de tipo *word*, es decir, del mismo tamaño que el registro CX del procesador. Se procede a la llamada de la interrupción 16 (servicios de la ROM BIOS), servicio 03H que devuelve, entre otros valores, la línea de exploración de comienzo del tamaño del cursor y la línea final, en los registros CH y CL respectivamente. Así, en la variable de la llamada se almacenará el tamaño del cursor que al final de la ejecución del programa servirá para recuperar el cursor con otra llamada a la interrupción 16 (procedimiento *visualizar_cursor*). También, en este procedimiento se llama al servicio 01H (establecer el tamaño del cursor) asignando el valor 20H al registro CX para poner el tamaño del cursor a cero (los números de líneas de exploración válidos ocupan sólo los cuatro bits de menor peso, por tanto el cursor desaparecerá).

Una vez que se ha llamado al programa, se debe comprobar si se ha introducido un parámetro o no, pasando las trayectorias adecuadas al procedimiento de búsqueda de archivos, éste es el cometido de la función *expandir*. Si no se ha introducido ningún parámetro, la trayectoria en la que se buscará será el directorio actual de la unidad por defecto, además se buscarán todos los archivos, por lo que la trayectoria será .*.*, si por el contrario, se ha introducido un parámetro, se comprobará si éste se corresponde con una unidad, es decir, el último carácter introducido en el parámetro es el carácter dos puntos, en cuyo caso se añadirá la trayectoria anterior. Una vez ajustado el parámetro se realiza una llamada a la función de la unidad *Dos*, *Fexpand*, que admite como parámetro una cadena de caracteres, devolviendo una trayectoria absoluta, es decir, compuesta por una unidad y la trayectoria completa desde el directorio raíz. Una vez que se ha construido perfectamente la trayectoria se llama al procedimiento *dividir_trayectoria*, que dada una cadena que contiene una trayectoria la divide en una trayec-

```
procedure localiza_archivos ( dir, arch: pathstr);
  var
   dir_sig: pathstr, { contiene el siguiente directorio a
buscar}
   dir_aux: pathstr, { contiene directorio\*.* para localizar
              todos los subdirectorios del directorio
              actual}
   area: searchrec; {contiene el DTA de cada archivo}
   ch_aux: char;
   lineas_ocupadas: integer;

  begin
   dir_aux:= dir + '\*.*';
   FindFirst(dir_aux, DIRECTORY, area);
   while (DosError <> 2) and (DosError <> 18)
     {no se han encontrado m s archivos}
     do
      begin
       if(((area.attr and DIRECTORY) = DIRECTORY)
         and (area.name[1] <> '.')) {si es directorio y es
                          distinto de los directorios
                          . y .. }
        then
         begin
          dir_sig:= dir + '\' + area.name;
          localiza_archivos (dir_sig, arch);
          { llamada recursiva para buscar dentro de
           los directorios encontrados en el directorio
           actual}
         end;
       FindNext(area)
      end;
{ una vez tratados todos los directorios del directorio actual
se procede a buscar todos los archivos del directorio actual}
   dir_aux:= dir + '\' + arch;
   FindFirst(dir_aux, $07, area);
   if DosError = 3 { trayectoria de directorio no encontrada}
    then
     writeln ('Trayectoria no encontrada')
    else
     while (DosError <> 2) and (DosError <> 18)
        do
         begin
          escribir_cabecera (dir, lineas_ocupadas);
          while (DosError <> 2) and (DosError <> 18)
              and (lineas_ocupadas < 22)
             do
             begin
              lineas_ocupadas:= lineas_ocupadas + 1;
              infor_archivo (area);
              FindNext (area);
             end;
          writeln;
          writeln ('Pulse una tecla para continuar... ');
          ch_aux:= readkey;
          clrscr;
          writeln ('Buscando...');
         end;
 end;
```

Cuadro 3.



toria de directorio más un especificador de archivo (nombre de archivo que puede contener caracteres comodín). Por último, se comprueba a través de la función *unidad_preparada*, si la unidad especificada en la trayectoria es correcta y funciona adecuadamente. Esta función comprueba que el primer carácter de la trayectoria de directorio sea una letra y si es así realiza una llamada a la función de la unidad *Dos*, *DiskSize*. Esta función no está diseñada para este propósito, pues su verdadero fin consiste en devolver el tamaño de la unidad especificada como parámetro, pero devuelve un valor negativo si se produce algún problema al acceder a la unidad. Se debe señalar, que esta función admite como parámetro un parámetro de tipo *byte*, siendo 0 la unidad por defecto, 1 para la unidad A y así sucesivamente, por lo que antes de realizar la llamada a esta función se ha procedido a transformar el carácter a *byte*, utilizando para ello la función *ord*, que devuelve el código ASCII de un carácter y aprovechando que las letras en el código son adyacentes.

En la función *unidad_preparada* se podría haber utilizado funciones proporcionadas por el DOS (función 3FH, leer desde un fichero o dispositivo) y posteriormente haber utilizado la función 59H (obtener información extendida de un error), pero para el fin perseguido es adecuado la utilización de dicha función.

```
Directorio de búsqueda:   C:\WINDOWS\SYSTEM

KBDSP     DLL          2.401 10/03/92    3:10  -H  -R  +A
LANGSPA   DLL          3.072 10/03/92    3:10  -H  -R  +A
LZEXPAND  DLL          9.936 10/03/92    3:10  -H  -R  +A
BWCC000C  DLL         89.536 20/04/94    8:52  -H  -R  +A
DDEML     DLL         36.864 10/03/92    3:10  -H  -R  +A
MMSYSTEM  DLL         63.456 10/03/92    3:10  -H  -R  +A
OLESVR    DLL         24.064 10/03/92    3:10  -H  -R  +A
SHELL     DLL         41.088 10/03/92    3:10  -H  -R  +A
WIN87EM   DLL         12.800 10/03/92    3:10  -H  -R  +A
PDX200    DLL        231.888 14/04/94    0:00  -H  -R  +A
TOOLHELP  DLL         14.128 10/03/92    3:10  -H  -R  +A
GENDRV    DLL         99.840 10/03/92    3:10  -H  -R  +A
TTEMBED   DLL        104.960 14/04/94    0:00  -H  -R  +A
SDM       DLL        102.032 14/04/94    0:00  -H  -R  +A
CTL3DV2   DLL         21.648 14/10/93    0:00  -H  -R  +A
VER       DLL          9.728 17/03/94    0:00  -H  -R  +A
SHARERES  DLL         37.888 15/12/93    0:00  -H  -R  +A
ODBCINST  DLL         83.856 12/11/93    0:00  -H  -R  +A
COMPOBJ   DLL        102.400 14/12/93    0:00  -H  -R  +A
OLE2CONV  DLL         57.328 14/12/93    0:00  -H  -R  +A

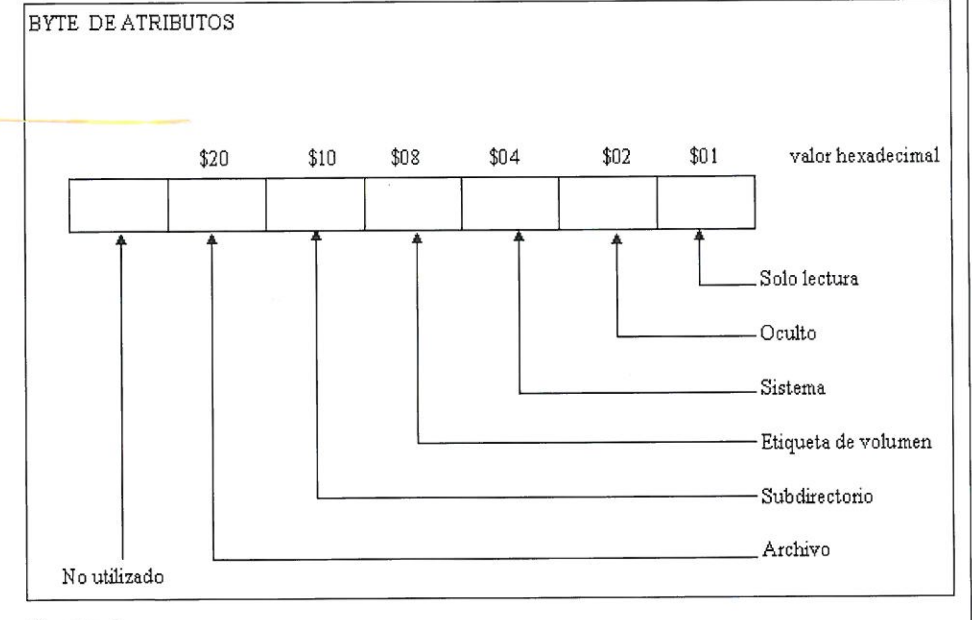
Pulse una tecla para continuar...
```

Cuadro 5.

EL PROCESO DE BÚSQUEDA

Una vez que se ha comprobado que la unidad está preparada, y que se dispone de una trayectoria y un descriptor de archivo correctos, se llama al procedimiento *localiza_archivos* que es el más importante de todo el programa (cuadro 3). Este procedimiento comienza asignando a una variable el especificador de archivos *.* junto al directorio donde se realizará la búsqueda, llamando posteriormente al procedimiento *FindFirst* junto con el valor *Directory* (se trata de una constante predefinida con valor $20 que se corresponde con el byte de atributos para el valor de directorio) en el parámetro correspondiente a los atributos, se está indicando que se desea encontrar un directorio dentro del directorio actual, antes de proceder al tratamiento del directorio debe comprobarse que la operación ha tenido éxito, *DosError* debe ser distinto de 2 (archivo no encontrado) y distinto de 18 (no hay más archivos que se correspondan con la especificación de archivo). Puesto que los directorios . (directorio actual) y .. (directorio padre) también existen como entrada, no se deben tratar cuando sean encontrados.

Una vez que se ha encontrado un directorio adecuado, se vuelve a llamar a la función *localiza_archivos* (llamada recursiva), pero ahora la trayectoria en la que se buscará será la trayectoria actualmente tratada más el nombre de directorio encontrado, es decir, se procederá a buscar en el directorio hijo, del



Cuadro 4.

directorio actual, los archivos que se corresponden con el especificador de archivo inicialmente buscado, que no varia. Este proceso continuará sucesivamente en cada uno de los subdirectorios que se encuentren en cada una de las llamadas. Una vez que se agoten los sucesivos subdirectorios, la llamada recursiva devolverá el control al proceso actual, que procederá a llamar al procedimiento *FindNext* para localizar, si los hubiera, otros subdirectorios. Como consecuencia, la visualización de los archivos que se encuentren en un directorio será posterior a la de los archivos encontrados en sus subdirectorios.

Una vez que se ha agotado la búsqueda en todos los subdirectorios de un directorio, queda el tratamiento de los archivos que se correspondan con el especificador de archivo introducido como parámetro del directorio que se está tratando. Puesto que ahora no se desean buscar los directorios, sino los archivos, habrá que modificar la búsqueda inicial. Así, se realiza una nueva llamada al procedimiento *FindFirst* pero ahora con la trayectoria actual (almacenada en la variable auxiliar *dir*) junto con el especificador de archivo e indicando que se desea localizar cualquier archivo sea cual sea su atributo (solo lectura, oculto, de sistema o de archivo), para este fin se introduce en el parámetro asociado a los atributos el valor $07 (ver cuadro 4).

De nuevo se comprobará que la función *FindFirst* se ha ejecutado correctamente, primero comprobando que el código de error es distinto de 3 (trayectoria de directorio no encontrada) y posteriormente volviendo a comprobar que no se dan los errores 2 y 18, que serán la condición para comprobar que no hay más archivos que buscar cuando se llame a la función *FindNext*. Aunque la condición de búsqueda parece más compleja en este trozo de código que en el correspondiente a la búsqueda de directorios, es similar, pues se ha complicado para presentar la información al usuario de forma adecuada, no para realizar la búsqueda.

Dentro del bucle dedicado a la búsqueda de archivos se realiza una llamada al procedimiento *infor_archivo*, que admite como parámetro la información del archivo encontrado actualmente, que se encuentra en el DTA representado por la variable *area*.

Como ya se comentó con anterioridad, la información que devuelve las funciones *FindFirst* y *FindNext*, no son adecuadas para su presentación en pantalla (salvo el nombre del archivo encontrado). El procedimiento *infor_archivo*, transforma la información obtenida por las anteriores funciones en información adecuada para el usuario en un formato similar al comando *Dir* del DOS (ver cuadro 5).

Así, se separan el nombre y la extensión, rellenándolos de blancos para que ocupen 8 y 3 caracteres, respectivamente y separando ambas cadenas por un carácter blanco. Posteriormente, se transforma el tamaño del archivo para que presente el punto como separador de miles. Puesto que la hora y la fecha de última modificación se encuentran empaquetadas en un entero largo, se desempaquetan mediante el procedimiento *UnpackTime* y posteriormente se separan con los caracteres *:* y */* justificándose con caracteres blancos o rellenando de ceros según sea el caso. Por último, se presenta la información sobre los atributos de los archivos que se han encontrado, siendo la notación utilizada similar a la notación que utiliza el comando del DOS *attrib* para visualizar los atributos de cada uno de los archivos.

El procedimiento *infor_archivo* puede mejorar su estructura, si se crean procedimientos y funciones locales a este procedimiento, a la manera de la función *atrib_cad* y que realicen cada uno de ellos la transformación de parte de la información que se desea representar, se deja al lector estas modificaciones.

El procedimiento infor_archivo puede mejorar su estructura, si se crean procedimientos y funciones locales a este procedimiento

CONCLUSIÓN

Como se ha visto a lo largo del artículo, los lenguajes de alto nivel proporcionan las suficientes herramientas para utilizar la información referente a la estructura de archivos utilizando la información facilitada por el sistema operativo, pero ocultando la complejidad que éstas pudieran tener. Sería interesante modificar el programa que acompaña a esta entrega, con el fin de poder realizar búsquedas mediante criterios múltiples, por ejemplo, poder incluir además del especificador de archivo, la fecha de última modificación o el tamaño del archivo.

COMPILACIÓN Y EJECUCIÓN DEL PROGRAMA

Para poder compilar el programa se necesitan las unidades *dos* y *crt* que acompañan al compilador de Turbo Pascal, versión 5.0 o superior.

Para ejecutar el programa una vez compilado, basta con introducir una trayectoria de directorio, junto con el nombre de un archivo, que puede contener caracteres comodín, obteniéndose los directorios y los nombres de archivo donde se encuentra dicho archivo.

CONCEPTOS Y HERRAMIENTAS UNIX PARA REDES TCP/IP

Fernando J. Echevarrieta

En el presente artículo, se plantea un alto en el camino para exponer y asentar algunos conceptos que permitan actuar con un mayor conocimiento de causa en la programación de comunicaciones y adentrarse en la administración de redes UNIX

A lo largo de los últimos meses se ha penetrado en el mundo de la programación de comunicaciones con una orientación fundamentalmente práctica, aunque se ha tenido siempre en cuenta la exposición previa de los conceptos necesarios para avanzar en cada tema conociendo el terreno sobre el que se pisaba. Este mes, la serie realiza una pausa en los aspectos de la programación para elevarse a un nivel desde el que contemplar como un todo las piezas del puzzle que se han ido desarrollando. Ya se han repasado diferentes mecanismos para comunicar aplicaciones y máquinas entre sí, aunque en próximos artículos se ampliarán los temas. Este artículo, alejándose del código, dejando en un nivel inferior la aplicación, toma una perspectiva mucho más amplia para observar el funcionamiento de Internet en su globalidad, desde un nivel de abstracción superior en el que apenas se distinguen las máquinas entre la maraña de redes. Eso sí, sin apartarse nunca del mundo UNIX.

REDES Y S.O. DISTRIBUIDOS

Se podría tomar como punto de partida para realizar una distinción entre redes y sistemas operativos, la definición que Tanembaum da de *red* como "colección interconectada de ordenadores autónomos". En este ámbito, la palabra *interconexión* significa en primer lugar, que los ordenadores se encuentran conectados entre sí, no importa por que medios: par trenzado, microondas, láser, coaxial, etc. En segundo lugar, se realiza un intercambio de información y, en tercer lugar, esta comunicación se lleva a cabo de igual a igual. Es decir, no existe una jerarquía como ocurría en los antiguos sistemas *mainframe-terminales*. En definitiva, cada ordenador dispone de su propio S.O. ejecutándose en su propia CPU.

La idea de red se contrapone a la de *S.O. distribuido*. Un S.O. distribuido elige por sí mismo las CPUs y los periféricos que va a emplear, todo ello en un esquema de interconexión transparente al usuario. En la medida en que esta transparencia sea mayor o menor, se tendrá una forma de cualificar el grado de distribución de un S.O.

Sistemas como UNIX pueden denominarse "S.O. de red" o "S.O. en red", pero no distribuidos, ya que el usuario debe indicar de forma explícita donde entrar, mover ficheros, enviar trabajos, etc. En definitiva, aunque el propio núcleo venga dotado de una API de comunicaciones será necesario asignar y dar nombre a las máquinas.

En una comunicación a través de una red se pueden distinguir dos tipos de elementos fundamentales: por una parte, los *sistemas finales* (*ES* o *End Systems*) que intervienen en la comunicación y, por otra, los *nodos intermedios* (IN) (figura 1) por los que pasa la comunicación.

FUNCIONAMIENTO DE INTERNET

Internet es, en realidad, una red virtual de conmutación de paquetes definida por software. Según Douglas E. Comer "un usuario piensa en Internet como una sóla red virtual que interconecta todos los hosts, y a través de la cual la comunicación se hace posible. Su arquitectura interna es, a la vez, oculta e irrelevante".

Su nombre proviene de que es realmente una red inter-red. Es decir, es una red de redes que no conecta entre sí ordenadores sino redes. Para Internet los nodos intermedios y los sistemas finales son redes de ordenadores, por lo que una vez un paquete llega a la red considerada como sistema final es responsabilidad de ésta hacerlo llegar a la máquina destino (figura 2). Esta concepción que a primera vista puede parecer una entelequia académica se verá reafirmada posteriormente en este artículo al estudiar el esquema de direccionamiento de Internet.

Cuando se habla de Sistemas Abiertos, como en esta serie, antes o después se habla del modelo *OSI* (*Open Systems Interconection*) de *ISO* (*International Standards Organization*). OSI es un modelo de referencia de comunicaciones que define 7 capas orientadas a resolver todos los problemas involucrados en una comunicación entre ordenadores. Internet no cumple este modelo ya que, de hecho, es diez años anterior a él. Sin embargo, en los niveles bajos, se puede encontrar cierto paralelismo, por lo que, a lo largo de estos artículos se ha empleado, y se continuará empleando terminología OSI al hablar de niveles. Aún así, la arquitectura *TCP/IP* de Internet, que es el tema que nos ocupa (el modelo OSI se apartará para otros artículos) se ha convertido en el estándar de facto de los Sistemas Abiertos, lo que no ha conseguido OSI (además de por razones técnicas, conseguir las especificaciones de un estándar OSI es difícil y caro y conseguir uno de Internet es tan simple como realizar un ftp gratuito). Esto es así ya que permite la comunicación entre sistemas heterogéneos independientemente de su arquitectura, sistema operativo, tipo de conexión física (nivel 1, físico) y configuración de red local (nivel 2, de enlace), ya sea *ethernet*, *token ring*, *token bus*, etc . Así pues, toda la estructura lógica de Internet se sustenta sobre el protocolo *IP* (*Internet Protocol*) tal y como se puede apreciar en la figura 3. Para que la comunicación sea posible, es suficiente con que la máquina que actúa en cada red como pasarela con Internet comprenda el protocolo. Incluso es posible montar IP sobre otros protocolos de red como *X.25*.

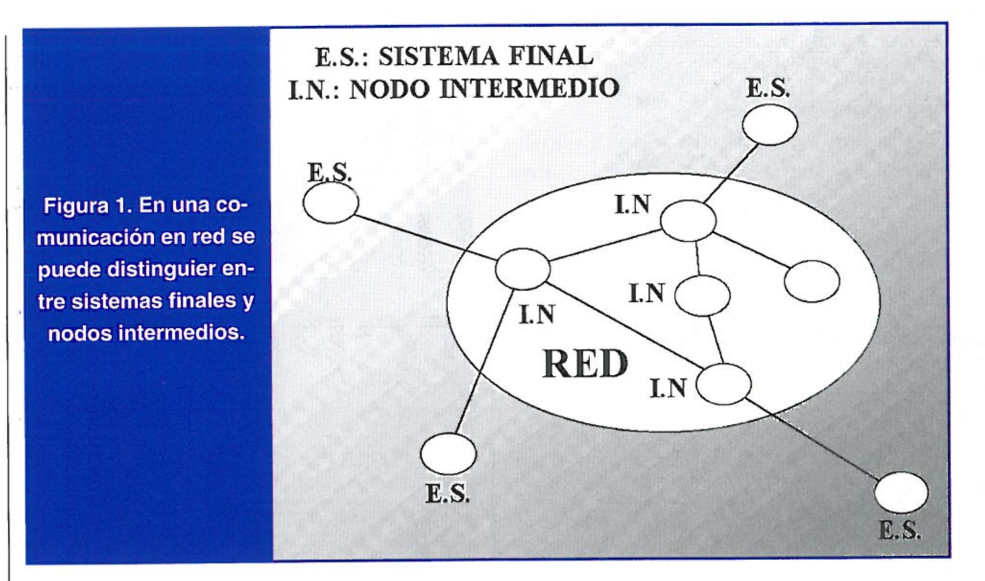


Figura 1. En una comunicación en red se puede distinguier entre sistemas finales y nodos intermedios.

CLASES DE DIRECCIONES

Al tratarse las direcciones IP de direcciones lógicas (en contraposición a las físicas, como las direcciones ethernet), se ha podido elegir la forma de definirlas del modo más conveniente para permitir un encaminamiento eficiente de la información. Desde este punto de vista, se pensó en las direcciones IP como estructuradas en dos campos: uno con información acerca de la *red* y otro con información sobre el *host* (máquina concreta). De este modo, se han estructurado las direcciones en *clases* como las que se pueden apreciar en la figura 5. A las tres primeras clases A, B y C, se les denomina direcciones clásicas, reservándose la clase D para IP multicast (multidestino) experimental, que no se tratará en este artículo, y la clase E para un hipotético uso futuro.

El procesamiento de estas direcciones es muy simple ya que bastarán los dos primeros bits para distinguir las direcciones de clase A, B o C. Así pues las direcciones clásicas se dividen en:

Clase A: Emplea 24 bits para identificar el host, por lo que se encuentra reservada para redes que puedan albergar más de 2 elevado a 16 máquinas, es decir, más de 65536 máquinas. Quedan 7 bits para la dirección de red, por lo que el límite máximo de redes en Internet con direcciones de clase A es 128.

Clase B: Emplea 16 bits para el host, por lo que se asigna a organizaciones que puedan administrar entre 2 elevado a 8 (256) y 2 elevado a 16 máquinas. Hay hasta 2 elevado a 14 posibles redes con direcciones de clase B.

Clase C: Emplea 8 bits para el host, por lo que se utiliza para designar redes de

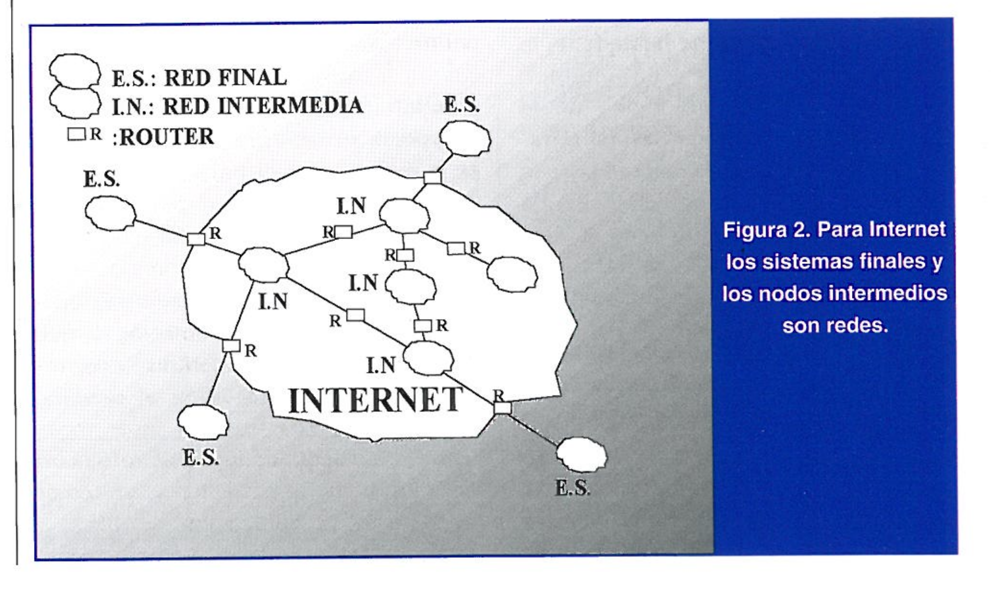


Figura 2. Para Internet los sistemas finales y los nodos intermedios son redes.

menos de 256 ordenadores. Hay hasta 2 elevado a 21 posibles redes con direcciones de clase C.

A lo largo del artículo se verá como, realmente, no todas las direcciones están permitidas ya que algunas tienen un uso especial reservado.

Con este tipo de direccionamiento, se codifica información sobre la red en las direcciones, lo que aporta como ventajas una gran eficiencia para el encaminamiento de la información así como la posibilidad de emplear las mismas direcciones para máquinas y para redes. Sin embargo, el rápido crecimiento de Internet y esta división en clases de sus direcciones están causando que la capacidad de direccionamiento de la actual versión de IP, *IPv4*, esté llegando al límite, ya que si una red cuenta con dos ordenadores habrá que asignarle una clase C, con lo que se desperdiciarán 254 direcciones. Por ello, la futura versión, *IPv6*, admite direcciones de 128 bits. Según Christian Huitema, ex-presidente del *Internet Architecture Board* e investigador del *INRIA*, esto podría admitir, calculando moderadamente, 1564 direcciones IP por cada metro cuadrado de la superficie de la tierra, aunque con otra reorganización menos pesimista se podrían llegar hasta más de un billón de direcciones por metro cuadrado. El objetivo sería que hasta el más simple casquillo de bombilla pudiera tener una dirección IP propia (Vin Cerf), lo que permitiría el control computerizado de todos los sistemas de la tierra.

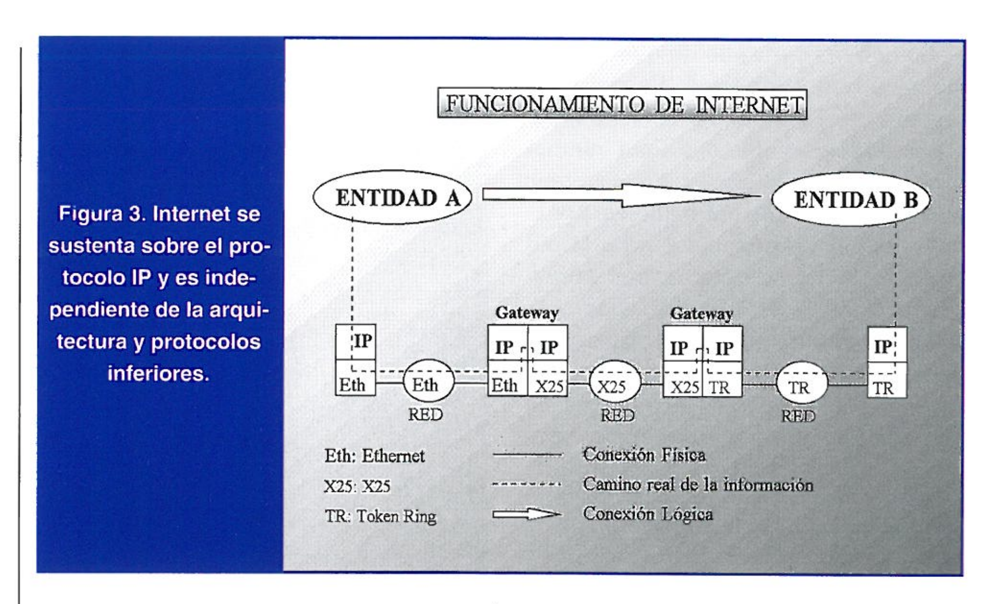


Figura 3. Internet se sustenta sobre el protocolo IP y es independiente de la arquitectura y protocolos inferiores.

SUBREDES

Hasta el momento, se ha hablado de la interconexión de redes en Internet, admitiendo el concepto lógico de la idea de red. Pero supóngase una organización que dispone de una red para lo que se le ha asignado una dirección de clase C, por ejemplo. Para Internet, esta organización será una única red lógica con su dirección correspondiente. Sin embargo, los administradores del sistema tendrán numerosos motivos para disponer internamente de varias redes físicas interconectadas entre sí. Entre estas razones destacan especialmente las de eficiencia y seguridad. Al separar el conjunto de ordenadores en varias redes físicas se produce una separación de tráfico, con lo que se evitan los problemas de saturación. Las redes pueden seguir una organización lógica por grupos de trabajo y, lo que es más importante, el tráfico entre máquinas de la misma red no sale al exterior, por lo que no se podrá "escuchar" desde otra subred, es decir, se habrá ganado en seguridad.

Por tanto, se puede ahora pensar en la dirección IP como integrada por dos partes: una parte para *Internet*, lo que antes se denominó información de red; y una parte *local*, lo que antes de denominó información sobre el host. Esta parte local ahora se puede subdividir de forma lógica nuevamente en información de la *subred* e información del *host*.

La ventaja de esto es que el administrador de cada red lógica conectada a Internet decide como particionarla. Retomando el ejemplo de la clase C, podría reservar 4 bits para designar la subred y otros 4 para el host, lo que permitiría hasta 16 subredes de hasta 16 máquinas cada una. Otra posibilidad sería reservar 2 bits para subredes y 6 para máquinas, lo que le permitiría hasta 4 subredes de hasta 64 máquinas cada una.

Para Internet únicamente habría una red correspondiente a la dirección de red de clase C asignada y, como se adelantó anteriormente, este sería el sistema final dentro de Internet. Una vez depositados en la máquina-pasarela hacia Internet, el destino de los paquetes sería responsabilidad de la propia red lógica, de la organización, que debería ser capaz de encaminarlos hasta la subred adecuada. Por ello, una vez se ha elegido una partición, todas las máquinas de la red deberán respetarla para lo que se define un nuevo elemento denominado *máscara de subred*.

MÁSCARAS DE SUBRED

Una máscara de subred (*subnet mask*) es un patrón de bits que se superpone a una dirección IP para obtener el número de subred a que pertenece esa dirección. Por tanto, deberá disponer también de 32 bits. Por ejemplo, en una clase C, los tres primeros bytes deberán siempre ser unos y será el último byte el que determine la partición. Así:

11111111 11111111 11111111 11110000

o, lo que es lo mismo

255.255.255.240

determinará una partición de hasta 16 posibles redes con 16 máquinas cada una como máximo. Del mismo modo, una máscara

255.255.255.192

determinará una partición de hasta 4 redes de 64 máquinas.

Para saber si dos máquinas se encuentran en la misma subred, se aplica la

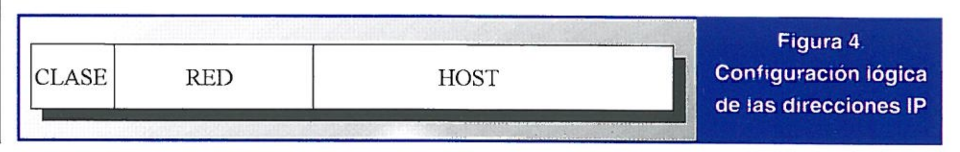


Figura 4. Configuración lógica de las direcciones IP

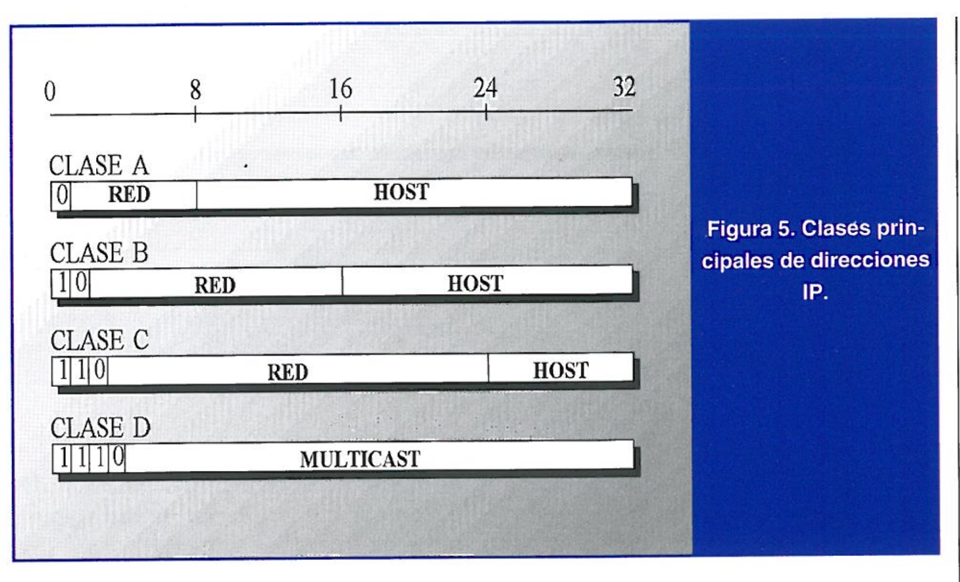


Figura 5. Clases principales de direcciones IP.

máscara de subred a ambas direcciones, de donde se obtiene la dirección de subred. Por ejemplo, supongamos una máscara 255.255.255.192 y las direcciones 138.4.2.10 y 138.4.2.76. Aplicando la máscara a la primera dirección:

138.004.002.010
255.255.255.192 &&

138.004.002.000

y con la segunda dirección:

138.004.002.076
255.255.255.192 &&

138.004.002.064

Por tanto, ambas máquinas se encuentran en subredes distintas, por lo que no podrán comunicarse directamente entre sí. Cuando esto ocurre, es necesario realizar la comunicación a través de una máquina que sirva como puerta de entrada y salida de la subred, por lo que recibe el nombre de *gateway* o pasarela de subred.

Como ejemplo para el lector, las siguientes líneas muestran un fragmento de un fichero */etc/networks* real (para mayor información sobre este fichero consúltese [Echeva-1]):

DITnet 138.4.2.0
CHINCHONnet 138.4.2.64
SIRIOnet 138.4.2.128
JUPITERnet 138.4.2.192

En el fragmento se puede apreciar una división en subredes. Se deja como ejercicio para el lector que determine a qué clase pertenecen estas direcciones y cuál es la máscara de subred empleada en la organización. El autor contestará con mucho gusto cualquier consulta que se le plantee.

REPETIDORES, BRIDGES, ROUTERS Y GATEWAYS

Otros términos que suelen aparecer, con bastante confusión, en administración de redes son los de repetidor, bridge, router y, de nuevo gateway. Los tres términos se emplean para designar máquinas o sistemas que sirven de nexo o puente entre redes y la diferencia radica en el nivel del modelo de referencia en que establecen esta unión.

Así, se denomina *repetidor* a una máquina o sistema físico que realiza un "empalme" entre dos segmentos de red. Es decir, se limita a reproducir las *señales* recibidas por el medio físico que tiene a un lado al que tiene al otro y viceversa sin "saber" realmente lo que está haciendo. Se trata de una conexión de *nivel 1, físico,* como pueda ser un radioenlace repetidor, un repetidor de televisión o una soldadura. Suelen ser de utilidad cuando surge una limitación por razones físicas, por ejemplo de atenuación de la señal. Un ejemplo de ésto es la máxima longitud de un cable de ethernet que se puede superar colocando un repetidor entre dos segmentos.

Un *bridge* o *puente*, por el contrario, es un sistema algo más "listo", que realiza una conexión en el *nivel 2, de enlace.* Esto suele ser de máxima utilidad cuando se dispone de dos redes con la misma capa de red, por ejemplo, IP, pero distinta capa de enlace. Por ejemplo, un conjunto de máquinas podría encontrarse conectado mediante ethernet y el otro mediante token bus. Al encontrarse en el nivel 2, un bridge entiende de direcciones de nivel de enlace, por lo que puede utilizarse como repetidor selectivo. Al leer las direcciones de enlace de destino de las *tramas* de información que le llegan por un lado, sabrá si es necesario repetirlas al otro o no en función de donde se encuentr la máquina destino.

En el *nivel 3, de red*, se encuentra el *router*, o *encaminador*, que recibe este nombre porque es el que se encarga de conducir los *paquetes* a una u otra red en función de su dirección de red de destino. Mediante los protocolos de encaminamiento, es capaz de decidir las rutas más adecuadas para este fin. De esta forma, es como se consigue que dos máquinas conectadas físicamente a dos redes distintas, puedan comunicarse entre sí, mediante la generación de una red virtual soportada por el nivel 3, lo que, en definitiva, es Internet.

El termino de *gateway* o *pasarela* es el que soporta una mayor ambigüedad. En unos casos, se aplica a cualquier unión que se establezca en un nivel superior al 3; en otros (quizá la aplicación más rigurosa), a aquella unión que se realiza entre redes heterogéneas en la que haya que realizar una conversión de protocolos; y en el caso más general, a la máquina que sirve como entrada o salida de una subred, como ya se ha indicado, o de una estructura lógica, en general, como pueda ser una organización.

Así pues, si bien en el diseño y planificación de redes se trata con todos estos elementos, para temas de administración y programación de comunicaciones a los niveles en que se está moviendo esta serie, se puede hablar siempre, de modo genérico de gateways.

DIRECCIONES ESPECIALES

Como ya se vio el mes pasado, existen algunas direcciones como la de *broadcast* o la de *loopback* que se encuentran reservadas y no se pueden asignar a un host o a una red. Así, por convenio, nunca se puede asignar el valor '0' a una dirección de host, por lo que aquellas direcciones con valor '0' en la parte del host designan redes. Del mismo

modo por convenio se asignó como dirección de broadcast aquella que ponía la parte de host toda a '1'. Sin embargo, debido a que las primeras implementaciones de BSD, tomaban como broadcast la parte de host toda a '0', hoy en día se pueden encontrar redes con cualquiera de los dos tipos de dirección como broadcast.

Respecto a estas direcciones, las hay de dos tipos: dirigidas y locales. Una dirección de broadcast *dirigida*, es aquella que consta de su parte de red y su parte local, siendo esta última la dirección de broadcast. Un paquete que lleve esta dirección como destino, alcanzará la red destino y allí será recibido por todas las máquinas que la compongan. Una dirección de broadcast *local* es aquella compuesta totalmente de '1', es decir, 255.255.255.255. Este último tipo de dirección solamente se permite durante el arranque, momento en que una máquina sin saber nada se dirige al mundo diciendo: "¿Hay alguien ahííííí...? ¿Quién soyyyyy?", esperando que alguien se lo comunique. Esto, que puede sonar a broma, es el desarrollo conceptual del protocolo *RARP* (*Reverse Address Resolution Protocol*), que no se tratará en esta artículo.

De este modo, el conjunto de direcciones especiales se puede resumir en:
Todo a '0' (0.0.0.0): Indica "yo" y se permite sólo al arrancar ("yo no sé quien soy y necesito que me lo digan")
Red a '0' + host: Indica "esta red" y se permite sólo al arrancar ("no se en qué red estoy y necesito que me lo digan")
Todo a '1' (255.255.255.255): Broadcast local. No se permite como dirección origen.
Red + host todo a '1': Broadcast dirigido
127 + cualquier cosa (generalmente 0.0.1): Loopback. La dirección de red 127 no está permitida.

COMANDOS BÁSICOS DE RED

Existe un conjunto de comandos básicos para la administración y gestión de redes en UNIX. Desgraciadamente, la sintaxis de los mismos puede variar con cada versión de este sistema e incluso algunos pueden cambiar de nombre o no encontrarse disponibles según versión. Por ello, lo más recomendable es hacer uso intensivo del manual en línea al cambiar de una versión a otra. En cualquier caso, a continuación se presentarán los fundamentos de su utilización así como las directrices para saber qué buscar en el manual. Los principales comandos de configuración de red son *ifconfig* y *route*, por lo que se verán en apartados específicos a continuación. Además existe una serie de útiles comandos como:
hostname: Se encarga de proporcionar o ajustar el nombre de la máquina.
netstat: Facilita información sobre las conexiones de la máquina. Sirve para trazar las conexiones activas, conocer los puertos de la máquina que están atendiendo, que máquinas se están conectando a qué servicios de la máquina local e, incluso, qué sockets de dominio unix [Echeva-2] se encuentran activos.
traceroute: Envía paquetes a una máquina y va informando de los nodos por los que pasan y los tiempos de tránsito. Si dispone de una conexión a Internet pruebe a ejecutar, por ejemplo: *traceroute highland.dit.upm.es*. Esta es una forma de detectar cuellos de botella y averiguar parte de la topología de la red.
ping: Envía un paquetes a la máquina indicada para averiguar si se encuentra accesible. Pruebe a ejecutar: *ping highland.dit.upm.es*
rpcinfo: Es un comando muy útil, especialmente para aquellos lectores que hayan seguido los artículos de programación mediante RPCs [Echeva-3], [Echeva-4]. Mediante el mismo es posible averiguar qué programas remotos sirve una máquina, qué versiones, en qué puerto atienden y sobre qué protocolos se sustentan. Para ello, basta teclear:
rpcinfo -p <máquina>
También permite llamar al procedimiento 0 de los mismos [Echeva-3].

Figura 6. División lógica de la parte local de una dirección IP para la partición en subredes

CLASE | RED | HOST
SUBRED | HOST

IFCONFIG

El comando *ifconfig*, se encarga de configurar los interfaces de red (generalmente tarjetas ethernet) con los parámetros adecuados, como la dirección IP del interfaz, máscara de subred y dirección de broadcast. En caso de emplearse sin parámetros muestra la configuración de todos los dispositivos disponibles en la máquina (en SunOs es necesario indicar la opción -a). En los listados 1 y 2 se puede ver el resultado de emplear el comando únicamente para obtener información. En el primer caso se puede apreciar cómo la máquina dispone de tres interfaces de red y en el segundo de dos (incluyendo el loopback). Aunque el formato cambia, la información que se facilita es prácticamente la misma, fundamentalmente: estado, dirección IP, broadcast y máscara de subred. Además se facilitan otros datos como *MTU* (*Maximun Transferable Unit*) o *Metric* (peso o métrica del enlace). La métrica es un parámetro de ponderación de la bondad del enlace para los algoritmos de encaminamiento que, en linux, por ejemplo, es ignorado.

En la práctica, se debe emplear *ifconfig* para configurar cada interfaz de red indicando su dirección IP, máscara de subred y dirección de broadcast y, en algunos casos es necesario activarlo (ponerlo en "UP"). Para ello, consúltese el manual en línea. El resto de la información puede ser omitida en la mayoría de los casos.

ROUTE

El comando route se encarga de mantener la tabla de encaminamiento de la máquina que consultará el software de comunicaciones cada vez que desee enviar un paquete. La tabla de encami-





LISTADO 1

Información proporcionada por ifconfig -a en SunOs

```
Echevarrieta:cuco>ifconfig -a
ie0: flags=863<UP,BROADCAST,NOTRAILERS,RUN-NING,MULTICAST>
inet 138.4.2.11 netmask ffffffc0 broast 138.4.2.0
ie2: flags=863<UP,BROADCAST,NOTRAILERS,RUN-NING,MULTICAST>
inet 138.4.1.129 netmask ffffffc0 broadcast 138.4.1.128
lo0: flags=849<UP,LOOPBACK,RUNNING,MULTICAST>
inet 127.0.0.1 netmask ff000000
```

LISTADO 2

Información proporcionada por ifconfig en Linux

```
root@highland:~> ifconfig
lo Link encap Local Loopback
inet addr 127.0.0.1 Bcast 127.255.255.255 Mask 255.0.0.0
UP BROADCAST LOOPBACK RUNNING MTU 2000 Metric 1
RX packets 0 errors 0 dropped 0 overruns 0
TX packets 38611 errors 4 dropped 0 overruns 0

eth0 Link encap 10Mbps Ethernet HWaddr 00:40:33:28:7C:60
inet addr 138.4.22.25 Bcast 138.4.22.0 Mask 255.255.255.192
UP BROADCAST NOTRAILERS RUNNING MTU 1500 Metric 1
RX packets 781781 errors 0 dropped 0 overruns 0
TX packets 923851 errors 0 dropped 0 overruns 0
```

namiento relaciona direcciones de destino con la dirección del router al que se debe dirigir el paquete para llegar a las mismas. Siempre existe una dirección por defecto. Así, por ejemplo, en el listado 3 se puede apreciar como para acceder a la redes 138.4.22.0 y 127.0.0.0 no es necesario emplear ningún gateway. Ya que la segunda es la dirección de loopback, se puede deducir que la máquina se encuentra en la red correspondiente a la primera dirección. Asímismo, se observa también que en caso de conocer como encaminar un paquete, se le envía a idit-gw1.dit.upm.es, que es la ruta por defecto. El comando proporciona mucha mas información, como el interfaz de red por el que se deben enviar los paquetes en cada caso, la métrica del enlace o la máscara de subred y sirve también para añadir o eliminar rutas a la tabla. Para mayor información, acuda al manual en línea.

CONFIGURACIÓN DE ARRANQUE DE RED

Al igual que ocurre con cualquier otro sistema operativo, una máquina cuando arranca, no sabe nada, ni su nombre, ni su dirección, ni su red, nada. Por ello, siempre existe una serie de ficheros de configuración para el arranque. La mejor forma de aprender labores de administración, si no de mantenimiento, al menos sí de configuración, es leer los ficheros de configuración del sistema que en UNIX, suelen ser siempre de lectura pública. La desventaja radica en que, como se ha mencionado, cada versión de UNIX tiene una forma de arranque distinta y los comandos de administración suelen disponer de opciones diferentes e, incluso, a veces, los propios comandos son diferentes.

LISTADO 3

Ejecución de route en Linux

```
Kernel routing table

Destination   Gateway        Genmask          Flags
Metric Ref   Use Iface
196.2.39.7      isabel.dit.upm.  255.255.255.255
UGHDM 0     0    464 eth0
ts900-3632.sing  isabel.dit.upm.  255.255.255.255
UGHDM 0     0    171 eth0
138.4.22.0 *  255.255.255.192 U  0  0  358497 eth0
127.0.0.0     *    255.0.0.0    U  0   0   31676 lo
default      idit-gw1.dit.up *  UG   0  0  93103 eth0
```

Suele ser habitual obtener los datos de nombre, dirección IP y gateway (o router por defecto) de sendos ficheros que leen los scripts, pero igualmente común es realizar una llamada *RARP* para obtener estos datos.

En lineas generales, se comentarán los ficheros de arranque de SunOs, UNIX tipo BSD y de Linux, tipo SYSTEM V, sistemas que se han venido tomando como referencia a lo largo de la serie.

En SunOs, los ficheros clave en el arranque son *rc* y *rc.boot*, ambos presentes en el directorio */etc* (modelo BSD) e invocados por el proceso *init* [Echeva-5]. Estos ficheros tienen como objeto realizar ciertas labores de mantenimiento en el arranque de la máquina así como arrancar los demonios [Echeva-5] adecuados y suelen ser iguales en todas las máquinas de una red. Además, suele encontrarse presente un fichero denominado *rc.local* en el que figuran aquellas labores que únicamente tiene sentido realizar en cada máquina concreta. Al arrancar, *init* ejecuta el fichero *rc.boot*, que suele ajustar el nombre de la máquina mediante *hostconfig* (un comando propio de SUN para consulta de información en red) y los parámetros de red mediante *ifconfig*. Si la máquina arranca en modo multiusuario, se ejecuta un *fsck -p* (*File System CheK*) que arregla las inconsistencias menores del sistema de ficheros y realiza un test para detectar otras más importantes. Si descubre un problema de importancia, *init* arrancará la máquina en modo monousuario.

En caso de llevarse a cabo el arranque multiusuario, se ejecutará el fichero *rc*. Este script se encarga de montar los sistemas de ficheros locales [Echeva-7] y ejecutar el *rc.local*. Éste último monta, a su vez, los sistemas de ficheros NFS, arranca los ficheros locales a la máquina y retorna, tras lo cual prosigue *rc* con el arranque de los demonios del sistema, preservación de ficheros de editores, borrado del */tmp*, comienzo de los accountings y configuración de red.

No obstante, se recuerda que estos ficheros de configuración son mantenidos por el root del sistema, por lo que estas acciones pueden variar en función de la configuración concreta de cada máquina.

En el caso de Linux, los ficheros de configuración para el arranque se encuentran en un directorio llamado */etc/rc.d* (modelo SYSTEM V). Allí, el primer fichero que se ejecuta en un arranque normal es el *rc.M* que, a su vez, inicializa el sistema de red llamando a los ficheros *rc.inet1* y *rc.inet2* del mismo directorio. El fichero *rc.inet1* realiza la configuración básica de la red llamando a *ifconfig* y *route* mientras que *rc.inet2* se encarga de lanzar los demonios de red.

PRÓXIMOS NÚMEROS

En futuros artículos se ampliará el conocimiento de la API de UNIX mediante funciónes de interacción con el kernel, se podrán algunos ejemplos de servidores de datagramas, se ampliará el tema de concurrencia, se contarán algunos "trucos" para la programación de servidores y se continuará con los temas de configuración de red y protocolos de Internet.

BIBLIOGRAFÍA

- Douglas E. Comer, "Internetworking with TCP/IP. Volume I: Principles, protocols, and architecture", 2nd Etition. Prentice Hall.
- Andrew S. Tanembaum, "Redes de Ordenadores", 2a Edición., Prentice Hall
- Olaf Kirch, "Linux Network Administrator's Guide", O'Reylly & Associates, Inc.
- Terry Dawson, "Linux NET2/NET3 HowTo"

Y, por supuesto, el manual en línea. Los dos últimos títulos se facilitaron en el CD-ROM de la Biblia del Linux entregado con el número 21 de la revista.

OBJETOS NATURAL

José María Peco

No hace mucho, un amigo del autor del presente artículo le contaba que había sido contratado por una empresa para mejorar la gestión de un almacén que fabrica 2000 puertas al día, lo cual, como es lógico, exige un cierto orden a la hora de gestionar los stocks para así optimizar el proceso de fabricación. Para poder llevar a cabo su misión, tuvo que empezar por convencer al encargado del almacén de que su cometido consistía en enseñarle a hacer las cosas de una nueva forma, pero sobre todo debió superar los recelos de éste, y el comentario siguiente: '*Esto lo he llevado yo 20 años, ahora no va a venir nadie a decirme cómo lo debo hacer*' a lo que mi amigo le contesto que una persona puede estar haciendo las cosas mal desde el principio, 1 , 20 y 60 años, sin que eso sea justificación de que las cosas se hacen bien.

Éste, que es un hecho real, también aparece, y muy a menudo, en el ámbito de los entornos de desarrollo informáticos, pues se debe tener presente que en estos entornos no llega al **5%** el personal que cuenta con titulación informática universitaria1, por lo que la gran mayoría de los desarrolladores se han autoformado, muchos de ellos sin poder preguntar a expertos ya que fueron enviados a su puesto de trabajo como *tales* por una empresa de servicio, y, con el tiempo crearon su propia '*escuela*'.

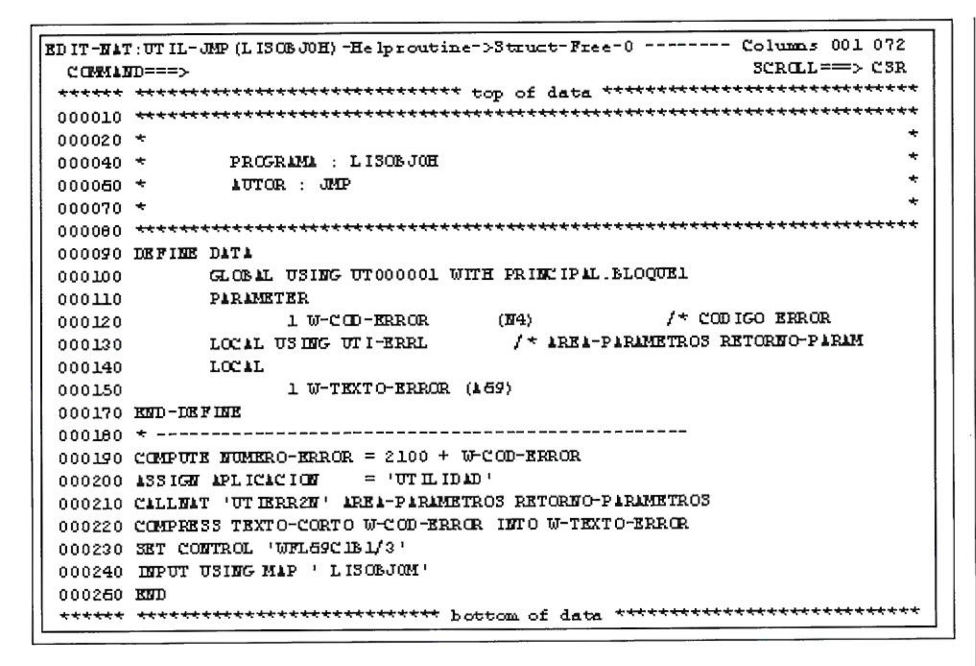
Este artículo se marca como objetivo el ayudar a los programadores de un entorno de desarrollo NATURAL a utilizar cada uno de los tipos de objetos de que dispone esta plataforma, para lo cual, el planteamiento que se va a seguir es definir cada uno de los tipos existentes y comentar los errores más frecuentes, pero, no se va a explicar ni la sintaxis ni los comandos de los editores usados para crear cada uno de ellos, ya que esto alargaría innecesariamente el artículo.

Todas las observaciones que se hagan en el presente artículo estarán referidas a la versión 2 de Natural, y más concretamente a NATURAL en modo estructurado, salvo indicación expresa en otro sentido.

La aparición de NATURAL en 1985 con su versión 1, supuso, al menos en España, una revolución en el mundo de los Grandes Sistemas, siendo superada tres años más tarde con la versión 2. El elemento diferenciador de este producto era un lenguaje de 4ª generación, que, entre otras muchas cosas favorecía la Modularidad

Grupo	Carácter asociado	Tipo
Áreas de Datos	G	- Global
	A	- Parameter
	L	- Local
Proceso	P	- Programas
	N	- Subprogramas
	S	- Subrutinas externas
Sistemas de ayuda	H	- Helptext
	H	- Helproutines
Otros	C	- Copycodes
	M	- Mapas
	T	- Texto
	D	- Descripción
	R	- Recording
DDM	V	- Modulo Definición Datos

Figura 1.



```
EDIT-NAT:UTIL-JMP(LISOBJOH)-Helproutine->Struct-Free-0 -------- Columns 001 072
  COMMAND===>                                                    SCROLL==> CSR
****** ****************************** top of data ****************************
000010 ***********************************************************************
000020 *                                                                     *
000040 *          PROGRAMA  : LISOBJOH                                       *
000060 *          AUTOR : JMP                                                *
000070 *                                                                     *
000080 ***********************************************************************
000090 DEFINE DATA
000100        GLOBAL USING UT000001 WITH PRINCIPAL.BLOQUE1
000110        PARAMETER
000120               1 W-COD-ERROR      (N4)             /* CODIGO ERROR
000130        LOCAL USING UTI-ERRL         /* AREA-PARAMETROS RETORNO-PARAM
000140        LOCAL
000150               1 W-TEXTO-ERROR (A69)
000170 END-DEFINE
000180 * -----------------------------------------------
000190 COMPUTE NUMERO-ERROR = 2100 + W-COD-ERROR
000200 ASSIGN APLICACION     = 'UTILIDAD'
000210 CALLNAT 'UTIERR2N' AREA-PARAMETROS RETORNO-PARAMETROS
000220 COMPRESS TEXTO-CORTO W-COD-ERROR INTO W-TEXTO-ERROR
000230 SET CONTROL 'WFL69CIB1/3'
000240 INPUT USING MAP ' LISOBJOM'
000260 END
****** *************************** bottom of data ****************************
```

Figura 2.

1.FUENTES Y OBJETOS

La arquitectura de NATURAL organiza, de cara al usuario, los programas en librerías, conteniendo cada una de ellas todos los módulos una aplicación[2]. Y, para todos aquellos módulos que son de uso común de la instalación, cuenta con una librería en la que, por defecto, el sistema buscará el módulo que no encuentre en la librería de su aplicación. Esta suele ser la librería **SYSTEM**, aunque puede tener cualquier otro nombre ya que este es asignado por el administrador de la instalación.

Dentro del entorno NATURAL, los continentes de los códigos fuentes y de los códigos ejecutables tienen el mismo nombre o identificador, siendo esto posible porque sus contenidos se almacenan en lugares distintos, gestionando el propio sistema este almacenamiento.

Los ficheros ADABAS[3] relacionados con esta función son:

FNAT : contiene los programas del sistema y de las utilidades.

FUSER : Contiene las librerías de aplicaciones FUENTES y OBJETOS.

FDIC : Contiene los Módulos de Definición de Datos (DDM).

Así, en un entorno NATURAL, se entiende por **FUENTE**, el código fuente escrito por el programador; y **OBJETO**, el código resultante de la compilación del fuente.

Los comandos del sistema que permiten manejar estos continentes cuando se tiene un fuente en el buffer del editor, son:

SAVE: el código fuente se guarda en el fichero del sistema.

CAT: el código fuente se compila, y el resultado de la compilación, si no ha habido errores, se almacena en el fichero del sistema. No se guarda el fuente.

STOW: el código fuente se compila, y el resultado de la compilación, si no ha habido errores, se almacena en el fichero del sistema, y además, en este caso, también se almacena el fuente. Equivale por tanto a **SAVE** + **CAT**.

Los comandos contrarios de estos son:

PURGE: Borra un fuente del fichero del sistema.

UNCAT: Borrar un objeto del fichero del sistema.

SCRATCH : Borrar fuente y objeto.

RUN: el código fuente se compila, y el resultado de la compilación, si no ha habido errores, se ejecuta. No se guarda el fuente ni el objeto.

2.TIPOS DE OBJETOS

La gran ventaja proporcionada por NATURAL en sus comienzos, junto a la de ser un lenguaje de 4ª generación, era la modularización que ofrecía, gracias a todos los tipos de objetos relacionados en la **figura 1.** Esto facilitaba la labor de programación, pues cambiaba los interminables programas COBOL por funciones elementales, capaces de resolver el mismo problema pero de un forma más simple y con menos código.

La **Figura 1** muestra los distintos tipos de módulos, también llamados *objetos*, clasificados por grupos, así como el carácter con el que generalmente se asocia el tipo. Este carácter suele ir, dependiendo de la nomenclatura asumida por la instalación, en una posición predeterminada, generalmente al final del nombre. Así, el nombre *JMP0000**G*** representa un área de datos Global, mientras que el *JMPXUT1**P*** representa el nombre de un objeto tipo programa.

Así mismo, se ha incluido el tipo de objeto **DDM** por ser muy usado por todos los programadores, aunque sólo pueda ser definible y modificable por el DBA[4]. Este tipo de objeto, ya tratado en el artículo dedicado a ADABAS, entre otras

```
UTIL-JMP JMPDES              Utilidad para desarrollo          18/04/96 15:17:25
                    Libreria: UTIL-JMP Objeto: LISOBJOP
-------------------------------------------------------------------------------
 0090 DEFINE DATA
 0100        GLOBAL USING UT000001 WITH PRINCIPAL.BLOQUE1
 0110        LOCAL
 0140   1 FUSER-D VIEW OF SYSTEM-FUSER
 0150     2 SRCID
 0160     2  REDEFINE SRCID
 0170        3 LIBRERIA         (A8)
 0180        3 PROGRAMA         (A8)
 0190        3 REGISTRO         (B2)
 0200     2 C*SRCTX
 0210     2 SRCTX               (1:2)
 0220 *
 0230   1 W-SRCTX              (A90)
 0240   1 REDEFINE W-SRCTX           /* <-- PRIMERA REDEFINICION
 0250     2 W-NUM-REG          (B2)
 0260     2 W-TEXTO            (A78)

      Proxima pantalla listar desde  280_     Buscar: ____________________
-------------------------------------------------------------------------------
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
                SALIR LOCAL MAPA
```

Figura 3.

```
15:10:19                    Define Map Settings for MAP                 96-04-18

 Delimiters                 Format                        Context
 -----------------          --------------------------    -------------------------
 Cls Att CD  Del            Page Size ...... 23           Device Check .... ________
  T   D      BLANK          Line Size ...... 100          WRITE Statement   _
  T   I      ?              Column Shift ... 0 (0/1)      INPUT Statement   X
  A   D      _              Layout ......... ________     Help ______________________
  A   I      )               dynamic ....... N (Y/N)       as field default N (Y/N)
  A   N      ^              Zero Print ..... N (Y/N)
  M   D      &              Case Default ... UC (UC/LC)
  M   I      :              Manual Skip .... N (Y/N)      Automatic Rule Rank  1
  O   D      +              Decimal Char ... ,            Profile Name .... SYSPROF
  O   I      (              Standard Keys .. Y (Y/N)
                            Justification .. L (L/R)      Filler Characters
                            Print Mode ..... __           -----------------------
                                                          Optional, Partial .... _
                            Control Var .... ________     Required, Partial .... _
                                                          Optional, Complete ... _
                                                          Required, Complete ... _

Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help        Exit                                                 Let
```

Figura 4.

cosas, se diferencia de los que se tratan aquí en que la DDM sólo define las características de los campos de un archivo o fichero de la Base de datos, es decir, no reserva memoria para ningún buffer, cosa que si hacen los distintos objetos agrupados bajo el titulo **ÁREA DE DATOS**, los cuales se basan en las características definidas en las DDMs para reservar el espacio correspondiente al buffer que servirá de intercambio de información con ADABAS. Otra diferencia se encuentra en el nombre, el cual puede tener hasta 32 caracteres, dando esto significado a su contenido.

3. CARACTERÍSTICAS DE LOS TIPOS

3.1 ÁREAS DE DATOS

Como el lector sabe muy bien, un **programa ejecutable**, con independencia del lenguaje en el que se haya escrito el fuente, no es más que la unión de los siguientes conjuntos:

- uno de posiciones de memoria, a las que se les asigna un nombre lógico para su más fácil manejo por el programador. Son las variables; y
- otro con la secuencia de instrucciones que debe ejecutar el procesador para manejar la información contenida en las posiciones de memoria definidas en el primer conjunto.

Pues bien, en el modo estructurado de Natural, cualquier objeto de tipo proceso debe especificar al principio del mismo las variables y los buffers que se van a usar en el programa, al igual que ocurría con la **Data División** de COBOL.

Estas variables, atendiendo al uso que se va a hacer de su contenido, pueden ser de dos tipos:

- Uso local : su contenido sólo puede ser utilizado por el módulo en el que se encuentra definida la variable y no se comparte su valor por los distintos ejecutables que contienen su definición.
- Uso Global: Su contenido es compartido por todos los programas de un usuario que contengan su definición y que se encuentren en la misma librería, de modo que una variable, puede cargar su valor en un programa para ser leído su contenido en otro.

Así pues, la definición de las variables locales, y/o de las estructuras se debe hacer dentro del programa y concretamente, al principio del mismo, tal y como muestra la **figura 2**, utilizando la sentencia *DEFINE DATA*. Pero, puede darse el caso de que un mismo conjunto de variables tenga que definirse en varios módulos ejecutables. Para evitar el tener que escribir en cada uno de esos módulos todas las definiciones, se puede usar el tipo de objeto **ÁREA de DATOS**, el cual no es un objeto ejecutable en el amplio sentido de la palabra, pero sus definiciones se incorporan al ejecutable en el momento de la compilación, con lo que se reduce el numero de líneas de fuente, tal y como se comprueba en el ejemplo de la **figura 2.**

Estas áreas de datos pueden ser de varios tipos, y, si bien la definición de las variables sigue el mismo mecanismo que si se definieran dentro del módulo, la diferencia fundamental entre ellas estriba en su disponibilidad en función del tipo de objeto que los utiliza.

- **Global** : contiene la definición de las variables globales, con la restricción de que sólo puede usarse un área global por programa. Y, como particularidad, tiene el hecho de agrupar en bloques las variables que contiene, de modo que sólo se reserve espacio en el programa para aquellas variables contenidas en los bloques que se especifiquen.
- **Local** : contiene la definición de variables que serán usadas en modo local dentro del programa. No tienen bloques, y las definiciones pueden ser de variables independientes, estructuras (o variables redefinidas) y definiciones de buffers asociados a vistas de ficheros. **Figura 2.**

```
Arr W-COD-ERROR                                                          Fmt N4
 ------------------------------------------------------------------------------
 AD= OILT______     ZP= OFF     SG= OFF    HE= 'LISOBJOH'__________     Rls 0
 NL= ______         CD= __      CV= ________________________________    Mod Data
 PM= __                         DY=
 EM= __________________________________________________________
 
 001    --010---+----+----+---030---+----+----+---050---+----+----+---070---+----
 (XXXXXXXXXXXXXXXXXXXXXX (XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX (XXXXXXX
 (XXXXXXXXXXXXXXXXXXXXXX (XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX (XXXXXXX
 ------------------------------------------------------------------------------
 SELEC      REFERENCIA          DIV            IMPORTE            TIPO   FECHA   ERR.
 ----- --------------------- --- -------------------- ------ -------- ----
 >:X   (MMMMMMMMMMMMMMMMMMMMM (XXX (MMMMMMMMMMMMMMMMMMMM (XXXXXX (XXXXXXXX.N999
 >:X   (MMMMMMMMMMMMMMMMMMMMM (XXX (MMMMMMMMMMMMMMMMMMMM (XXXXXX (XXXXXXXX (9999
 >:X   (MMMMMMMMMMMMMMMMMMMMM (XXX (MMMMMMMMMMMMMMMMMMMM (XXXXXX (XXXXXXXX (9999
 >:X   (MMMMMMMMMMMMMMMMMMMMM (XXX (MMMMMMMMMMMMMMMMMMMM (XXXXXX (XXXXXXXX (9999
 >:X   (MMMMMMMMMMMMMMMMMMMMM (XXX (MMMMMMMMMMMMMMMMMMMM (XXXXXX (XXXXXXXX (9999
 >:X   (MMMMMMMMMMMMMMMMMMMMM (XXX (MMMMMMMMMMMMMMMMMMMM (XXXXXX (XXXXXXXX (9999
 >:X   (MMMMMMMMMMMMMMMMMMMMM (XXX (MMMMMMMMMMMMMMMMMMMM (XXXXXX (XXXXXXXX (9999
 >:X   (MMMMMMMMMMMMMMMMMMMMM (XXX (MMMMMMMMMMMMMMMMMMMM (XXXXXX (XXXXXXXX (9999
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      HELP  Mset  Exit  <---  --->  --    -     +           <     >     Let
```

Figura 5.

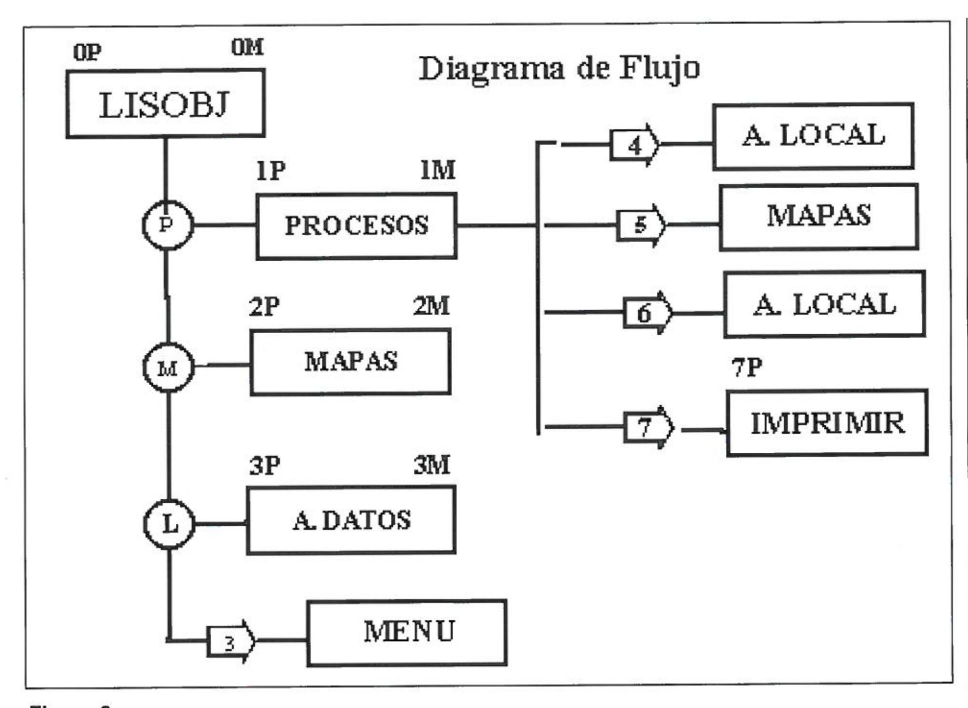


Figura 6.

• **Parameter**: Este tipo de área sólo puede ser usada en objetos del tipo subrutina y subprograma. Y contiene la definición de las variables que serán recogidas como parámetros iniciales en el programa o función llamado. Un tipo especial de variable es aquella que referencia campos de un fichero. En este sentido, cabe recordar que Natural, es un lenguaje íntimamente relacionado con el SGDB **ADABAS**[5]%ANOT≠¡®˘JMP. Los ficheros son definidos físicamente por el administrador de la Base de datos (**DBA**), pero son gestionados por ADABAS. Para que el programador pueda grabar o recuperar información de estos ficheros, el administrador, (que es una persona del departamento de sistemas[6]) pone a su disposición unos módulos denominados **DDM** (Definition Data Module). En ellos se encuentran definidos sólo los nombres y atributos de aquellos campos del fichero que pueden ser vistos por el programador, con lo que se esta restringiendo el acceso a datos confidenciales. Evidentemente, puede haber DDMs que contengan las definiciones de todos los campos de un fichero, y restringir el acceso a la información de determinados campos mediante el uso de ***NATURAL SECURITY*** (otra utilidad proporcionada por el sistema NATURAL que gestiona y controla las autorizaciones de acceso).

Pues bien, el buffer o posiciones de memoria que servirán de tampón intermediario para enviar y/o recibir datos de la base, también se debe encontrar definido dentro de cualquiera de los tipos de áreas especificados, aunque lo normal es que se defina dentro de un área local, y no dentro de una global, ni que se pase como parámetro a otro objeto.

Esta estructura cuenta con la siguiente particularidad : No necesita definir los atributos de formato y longitud de cada campo, pues asume los que se encuentren definidos en la DDM, aunque en caso de definirlas, esta afectaría sólo a su longitud, no a su tipo; y además, no necesita definir todos los campos de la DDM, sólo aquellos campos que se vayan a usar dentro del programa. Por esta razón se denomina **VISTA (view)**. Entre las recomendaciones dadas por **SOFTWARE AG**[7], se encuentra el aconsejar el uso de **view internas**, tal y como muestra la **figura 3** ya que esto reduce el tamaño del objeto, y además aumenta la eficiencia de **ADABAS** ya que este gestor debe generar un *IFB* (Internal Format Buffer) por cada campo de la vista, lo cual es muy costoso en consumo de CPU.

3.2 PROGRAMAS

Con independencia del concepto de programa visto al principio del apartado anterior, se entiende por *objeto tipo programa* un proceso constituido por un conjunto de instrucciones que se pueden ejecutar independientemente de otros objetos. En este sentido, un objeto tipo programa, una vez compilado, pasa a denominarse **comando**, y en consecuencia puede invocarse exactamente igual que cualquier otro comando del sistema.

La instrucción Natural que provoca que se ejecute un programa es la sentencia *FETCH* o *FETCH RETURN*, aunque también puede ejecutarse cargando en el stack, o pila de comandos pendientes de ejecución, el nombre del programa a ejecutar mediante la sentencia:

STACK [TOP] COMMAND nombre_pgm

Cuando Natural detiene la ejecución de instrucciones, bien porque se haya ejecutado la sentencia **STOP**, bien porque haya terminado de ejecutar las sentencias de un programa, lo primero que hace NATURAL es examinar el **stack** para saber si tiene que ejecutar algún comando, y, si es así, carga el nuevo ejecutable y comienza la ejecución de sus sentencias. Por último, en este punto cabe resaltar que aunque se pueda ejecutar un programa vía *FETCH RETURN*, se aconseja usar los tipos de módulos subprogramas y subrutinas, y reservar los módulos tipo programa para los procesos de tipo programa principal.

3.3 SUBRUTINAS

Estas pueden ser **internas** y **externas**. Las subrutinas internas se encuentran definidas en el mismo módulo que el programa llamante, por lo que_no requieren el paso de datos. Se corresponden con un conjunto de instrucciones agrupadas bajo un nombre, formando una rutina interna, de modo que pueda ejecutarse de 0 a n veces. Comparte todas las variables con éste, y su nombre puede tener hasta 32 caracteres, lo cual añade significado. (Ejemplo: *Perform PASAR-FECHA-AAAAMMDD-A-DDMMAAAA*).

En el caso de subrutinas externas, estas se encuentran definidas en un módulo distinto al del proceso llamante. Este módulo normalmente tiene por nombre el de la subrutina, aunque hay que tener presente que un módulo *tipo subroutine* puede contener varias subrutinas externas, por lo que se debe diferenciar entre nombre del módulo y nombre de la subrutina, y recordar que se invoca una subrutina externa por su nombre y no por el nombre del módulo que la contiene.

WATCOM C++ 10.5: EL COMPILADOR DE MODA

Fernando de la Villa

00101010010
00101010010
10101001010
01010101001
01010101010
101000100111
10101010101
00101101010
110101101010

Watcom C++ es un compilador que goza de una gran aceptación por parte de los desarrolladores de software debido, entre otras cosas, a su calidad y velocidad. Si a esto se le añade un nutrido conjunto de herramientas y una gran versatilidad se obtiene una herramienta de desarrollo de lo más recomendable

Watcom C++ es una herramienta multiplataforma de desarrollo para ordenadores PC que ha sabido abrirse paso en el mercado por méritos propios. Se trata de un potente compilador de los lenguajes C y C++ muy versátil, completo y dotado de una correcta documentación, que se ha hecho famoso por su capacidad para generar código de gran calidad. La versión 10.5, sobre la que se centra el presente artículo, incluye un gran número de utilidades y permite la creación de aplicaciones tanto de 16 como de 32 bits para las plataformas más utilizadas en el entorno PC.

CARACTERÍSTICAS

Una de las mayores ventajas de este paquete es la capacidad de desarrollo multiplataforma. Con un mismo código fuente se pueden crear ejecutables destinados a distintos entornos operativos, con el ahorro de tiempo y dinero que esto supone. Las plataformas soportadas por Watcom C++ son las siguientes:
- DOS (16 bits).
- Windows 3.x (16 bits).
- OS/2 1.x (16 bits).
- DOS extendido (32 bits).
- Windows 3.x (con la tecnología de extensor 32 bits de Watcom).
- Win32s (32 bits).
- Windows 95 (32 bits).
- Windows NT (32 bits).
- OS/2 (32 bits).
- NLMs de Novell (32 bits).
- AutoCAD ADS (32 bits).

Además, el programador puede elegir la plataforma de desarrollo que resulte más adecuada a sus necesidades. Watcom C++ incluye un Entorno Integrado de Desarrollo (EID) para facilitar el trabajo, pero siempre permite la utilización de la línea de comandos. Sin embargo, el EID (IDE en inglés) no está disponible para todas las plataformas. El compilador puede ejecutarse en los siguientes entornos:
- DOS (sólo línea de comandos).
- OS/2 32 bits (EID y línea de comandos).
- Windows 3.x (EID).
- Windows 95 (EID y línea de comandos).
- Windows NT (EID y línea de comandos).

Watcom C++ incluye un depurador (debugger) muy completo y mejorado frente a las versiones anteriores del compilador. Esta versión proporciona versiones gráficas del depurador para Windows 3.x/95/NT y OS/2 de 32 bits, además de versiones en modo texto para DOS, Windows 3.x, Windows NT y OS/2 de 32 bits.

Con el fin de facilitar la creación de interfaces gráficos de usuario, también se incluye la herramienta Visual Programmer de Blue Sky Software. Se trata de un programa ejecutable exclusivamente bajo Windows que simplifica la tediosa tarea de diseñar el aspecto de las aplicaciones Windows. Visual Programmer proporciona un área de diseño en la que se van colocando los distintos controles del interfaz de usuario Windows hasta obtener los resultados deseados. En ese momento se genera automáticamente el código fuente necesario y se puede incluir en el proyecto que se está desarrollando.

En esta versión se ha incluido soporte de las Microsoft Foundation Classes (MFC) 3.0 para aplicaciones 32 bits bajo Windows NT, Windows 95 y Win32s, y además se soportan las MFC 2.5 para aplicaciones 16 bits bajo Windows 3.x. También se han incluido componentes de varios kits de desarrollo (SDK) y toolkits como las librerías API de Windows 3.1, las APIs del Presentation

Manager de Os/2 2.1, y los archivos de cabecera necesarios para crear un NLM de Novell.

El paquete dispone del conocido extensor DOS de 32 bits DOS/4GW, muy utilizado en software de tipo lúdico. Este extensor permite utilizar 32 MB de memoria saltándose la limitación de los 640 KB de memoria base, y además está libre de royalties. Pero también se soportan otros extensores de DOS como el TNT de Phar Lap y el extensor de FlashTek.

Watcom C++ también incluye un ensamblador compatible con un subconjunto del ensamblador de Microsoft que puede utilizarse desde la línea de comando. Y como no podía ser de otra forma, se han incluido diversos ejemplos de aplicaciones para distintas plataformas que demuestran las posibilidades del compilador y del Entorno Integrado de Desarrollo.

Además, se proporcionan editores textos para la creación de código fuente, diversas utilidades de desarrollo multiplataforma, un navegador de clases y herramientas de análisis del rendimiento de las aplicaciones como el Watcom Execution Sampler y el Watcom Execution Profiler.

HERRAMIENTAS VISUALES

Para el desarrollo de aplicaciones Windows el paquete de Watcom C++ incluye una serie de herramientas visuales que sirven de perfecto complemento al compilador. Las ventajas que proporcionan estas herramientas son claras, pero no pueden ser aprovechadas más que en Windows 3.x, Windows 95 y Windows NT, a excepción de los editores de recursos para OS/2 incluidos en la distribución

A continuación se describen brevemente cada una de estas herramientas visuales:

- Resource Editor: estas utilidades se emplean para la creación y edición de recursos que posteriormente serán utilizados en las aplicaciones tanto de 16 como de 32 bits. Los aceleradores, mapas de bits (imágenes bitmap), cursores, cuadros de diálogo, iconos, menús y cadenas son recursos. Mediante el editor de recursos se puede modificar fácilmente el interfaz de la aplicación sin necesidad de tocar una sola línea de código C o C++.

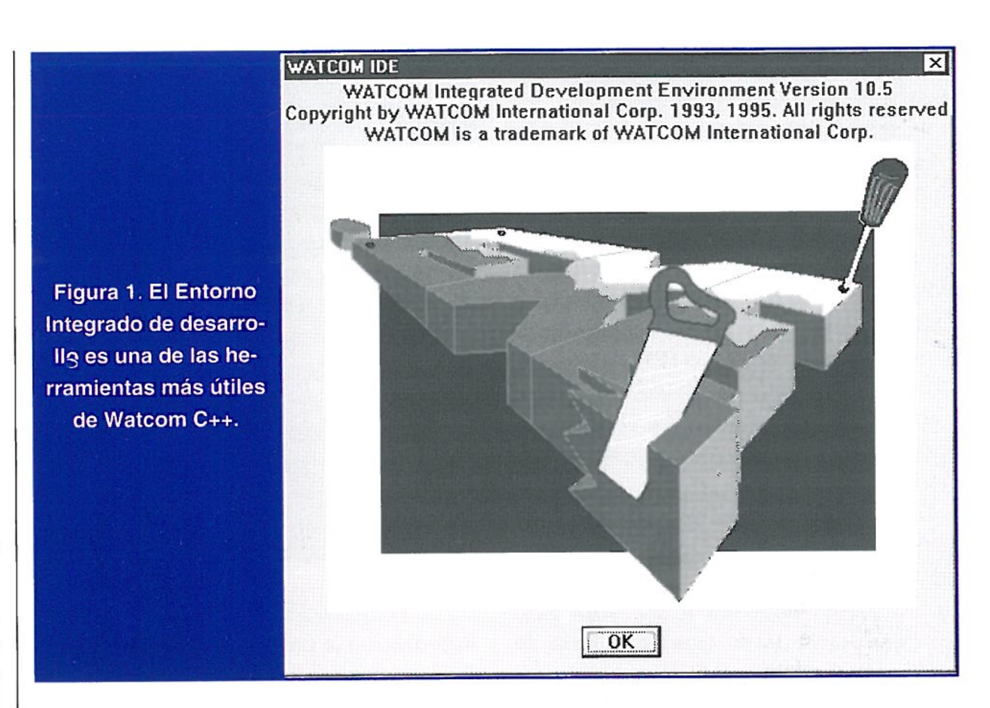


Figura 1. El Entorno Integrado de desarrollo es una de las herramientas más útiles de Watcom C++.

- Resource Compiler: esta no es una herramienta visual propiamente dicha, pero está directamente relacionada con la programación visual de Watcom C++. Se trata de una utilidad que funciona desde la línea de comandos y que tiene dos funciones principales. La primera es la conversión de archivos fuente de recursos (archivos con la extensión .RC) en archivos de recursos (archivos con la extensión .RES). La segunda función es la integración de estos archivos de recursos con archivos ejecutables o librerías de enlace dinámico (DLLs). Esta versión del compilador de recursos solamente soporta recursos de Windows y Windows NT.
- Zoom: esta utilidad, como su propio nombre indica, sirve para ampliar porciones de la pantalla. A la hora de desarrollar una aplicación se suele trabajar con resoluciones de 1024x768 puntos o superiores, y en ocasiones es difícil ver a simple vista un error en los gráficos o en la disposición de recursos de una aplicación. Con la herramienta zoom se puede examinar con detalle el interfaz gráfico de nuestra aplicación y subsanar los errores que en otras resoluciones no sólo se detectarían con mayor facilidad, sino que resultarían más molestos para el usuario final.
- Heap Walker: se trata de una valiosa herramienta de depuración cuya finalidad es la de supervisar el uso de la memoria y examinar el contenido de la misma. Heap Walker ayuda a encontrar datos corruptos, problemas de memoria y usos ineficientes. También permite la modificación de los contenidos de una zona de memoria para la realización de

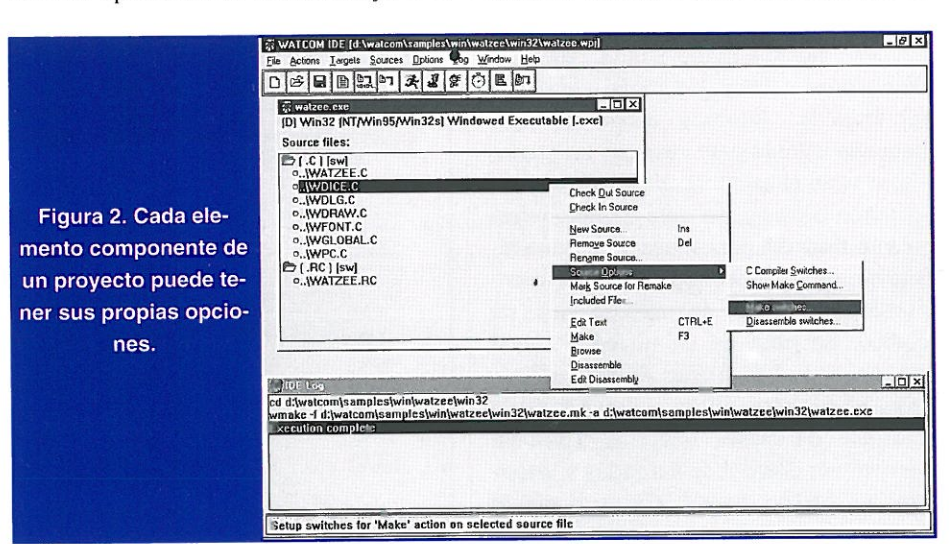


Figura 2. Cada elemento componente de un proyecto puede tener sus propias opciones.

pruebas. Bajo Windows NT sólo permite examinar la memoria utilizada por las aplicaciones Windows.

- Spy: el espía es otra herramienta con fines de depuración de programas. Con Spy se pueden examinar los diversos mensajes entre la aplicación y Windows, detectando así la parte del programa que está actuando de forma errónea. Los programadores menos experimentados también pueden utilizar esta herramienta para aprender en la práctica el mecanismo de mensajes utilizado por los entornos Windows.
- DDESpy: el Intercambio Dinámico de Datos (Dynamic Data Exchange) es uno de los métodos de comunicación entre aplicaciones disponibles en Windows. DDESpy sirve para examinar toda la información relativa al DDE que se produce en el sistema. También es posible la visualización de errores DDEML y el registro de varios objetos DDE.
- Dr. Watcom: sirve para analizar las posibles causas de la generación de una excepción. Doctor Watcom es una herramienta que facilita el estado de la aplicación y del propio sistema para detectar el origen del problema. También permite visualizar y modificar la información sobre los procesos.
- Image Editor: esta herramienta se utiliza para crear recursos gráficos como imágenes, mapas de bits, iconos y cursores. El Image Editor presenta el aspecto típico de cualquier aplicación de dibujo para Windows y aunque no dispone de unas opciones avanzadas, cumple con su cometido. Tiene herramientas de selección, de desplazamiento de imagen, de rotación, de inversión y de retoque de colores. Las herramientas de dibujo incluyen dibujo a mano alzada, rellenado de regiones y dibujo de líneas, círculos y rectángulos. Además incorpora una pequeña utilidad para capturar imágenes situadas en el escritorio de Windows.
- Source Browser: se trata de una herramienta muy útil para realizar el mantenimiento de grandes aplicaciones compuestas por cientos de miles de líneas de código. Se trata de un navegador que estudia el código fuente de una aplicación y muestra información como la jerarquía de clases utilizada. También permite ver el árbol de llamadas y visualizar el código fuente correspondiente perteneciente a un elemento determinado de la aplicación. La mayor parte de la información la presenta de forma gráfica, facilitando de este modo el seguimiento del código. Sirve tanto para recordar la estructura de una aplicación creada hace algún tiempo por el propio programador y también para comprender el funcionamiento de un programa creado por terceras personas.

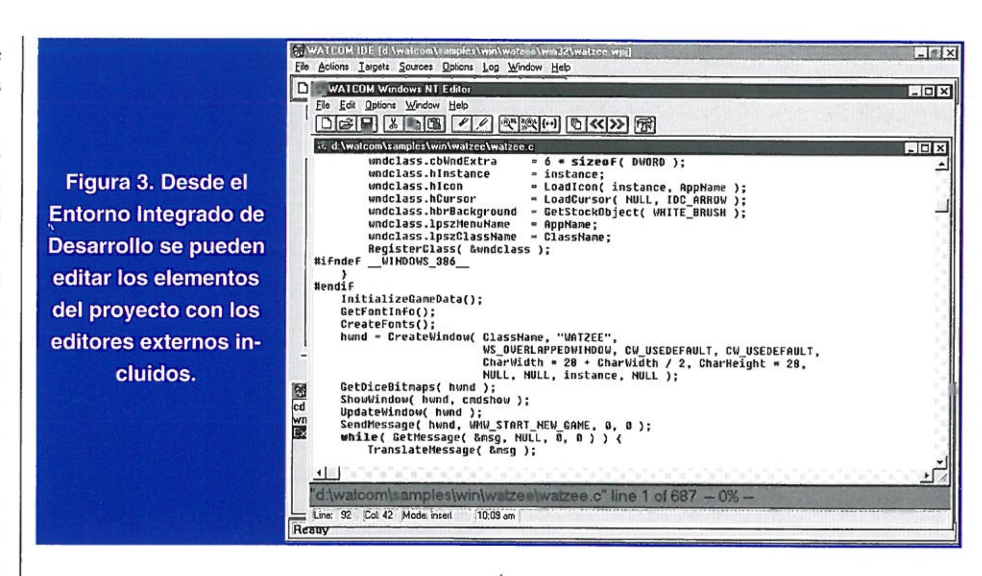


Figura 3. Desde el Entorno Integrado de Desarrollo se pueden editar los elementos del proyecto con los editores externos incluidos.

DOCUMENTACIÓN ON-LINE

Otro de los aspectos destacados de Watcom C++ es la documentación on-line adjunta. Este tipo de documentación no es el más adecuado para aprender a utilizar el producto desde el principio, pero sin duda alguna es el ideal para trabajar. El hecho de disponer de toda la información en cómodos archivos de ayuda facilita la programación y ahorra mucho tiempo de búsqueda en las páginas de un libro.

Estos archivos de ayuda están organizados en dos grandes grupos: ayuda adicional y ayuda sobre las herramientas. El primero contiene archivos de ayuda útiles a la hora de programar. Existe ayuda sobre las Microsoft Foundation Classes, los comandos MCI, multimedia y el API Pen, entre otras. El segundo grupo contiene la información de todas las herramientas incluidas en el paquete, además de las guías de referencia del lenguaje C y del C++, las librerías de ambos lenguajes, la guía del usuario y la guía del programador. Por último, se incluye una guía de referencia rápida que enseña los primeros pasos con Watcom C++.

La mayor parte de estos archivos están en formato ayuda de Windows. Sin embargo, las ayudas sobre el API de Win32s y las MFC están sacadas de sus respectivos kits de desarrollo y utilizan un formato propio que se visualiza con una utilidad específica de Microsoft.

Si de todos modos se quiere disponer de la documentación impresa, las guías de Watcom C++ y las guías de

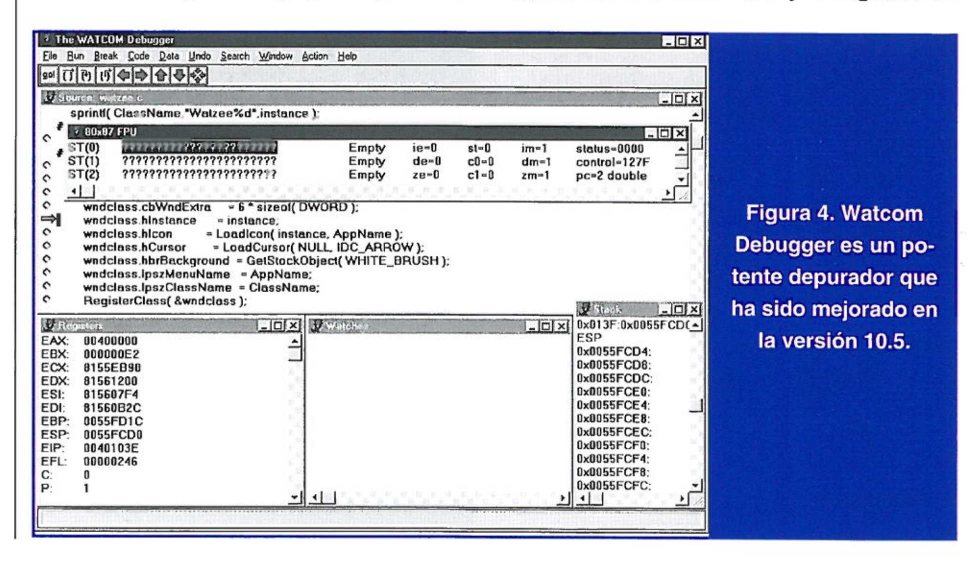


Figura 4. Watcom Debugger es un potente depurador que ha sido mejorado en la versión 10.5.

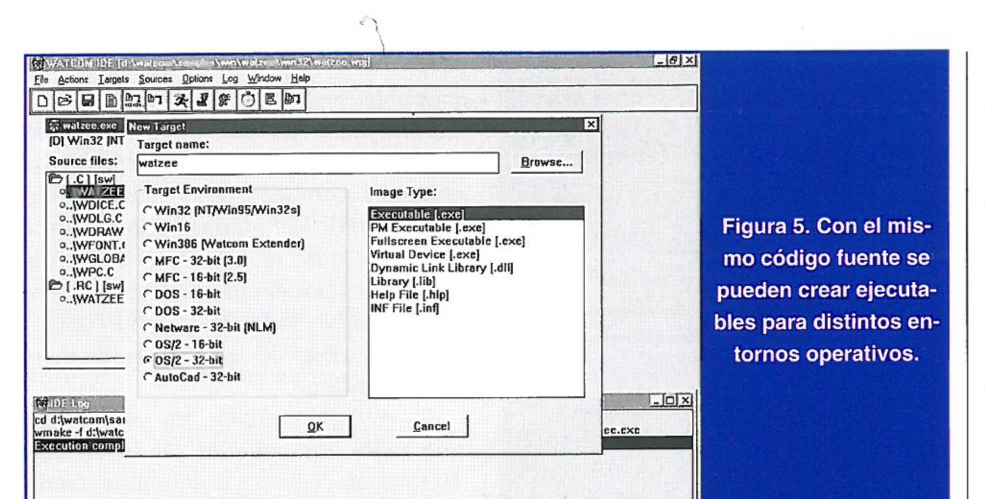


Figura 5. Con el mismo código fuente se pueden crear ejecutables para distintos entornos operativos.

referencia pueden encargarse directamente a Watcom.

INSTALACIÓN

Para poder ejecutar Watcom C++ se necesita un ordenador compatible IBM PC con las siguientes características:

- Procesador 386 o superior.
- 8 megas de memoria RAM.
- Unidad de CD-ROM.

El compilador también puede ser instalado en ordenadores que no dispongan de unidad de CD-ROM. La instalación de Watcom C++ se puede transferir desde el CD de la distribución a 127 discos de alta densidad, con lo que queda resuelto el problema.

El espacio libre necesario en el disco duro depende del tipo de instalación. La instalación completa ocupa un total de 385 megas y la mínima (ejecutando desde CD) ocupa aproximadamente 4 megas.

Además de los requisitos ya citados, se necesita disponer de al menos uno de los sistemas indicados a continuación:
- DOS 5.0 o superior.
- Windows 3.1 en modo mejorado (enhanced mode).
- Windows 95.
- Windows NT 3.1 o superior.
- IBM OS/2 2.1 o superior.

La instalación de la versión 10.5 ha sido mejorada considerablemente respecto a versiones anteriores. Ahora el programa de instalación es más inteligente y permite realizar una instalación sobre una versión más antigua con mayor facilidad. De forma automática se examinarán los componentes instalados y se actualizará a la nueva versión conservando la configuración existente. También se permite una instalación incremental, es decir, el programador puede instalar componentes a medida que los vaya necesitando y puede eliminar del disco duro los que no utilice.

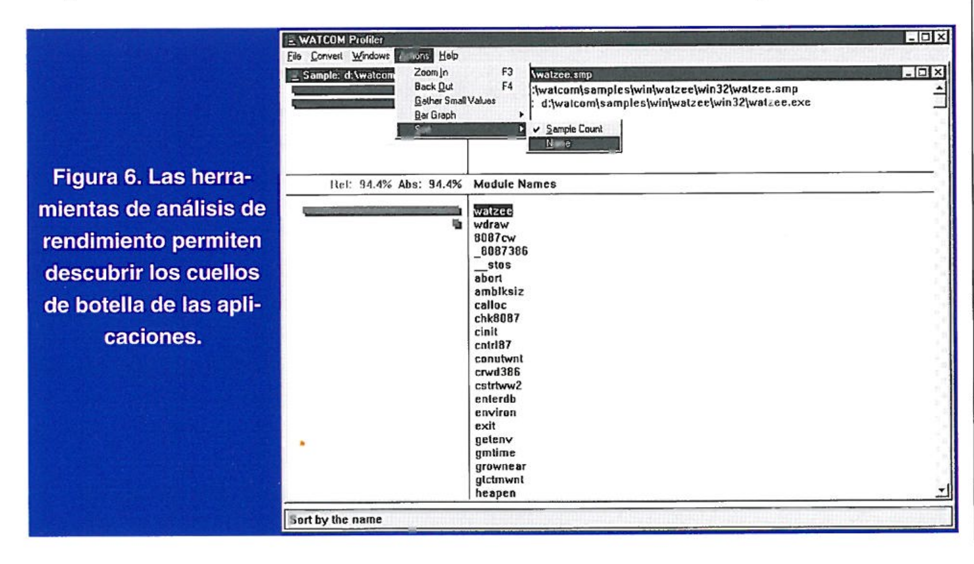


Figura 6. Las herramientas de análisis de rendimiento permiten descubrir los cuellos de botella de las aplicaciones.

Bajo Windows se crean tres grupos de programas diferentes con el fin de estructurar la gran cantidad de iconos necesarios. El grupo principal contiene los programas ejecutables y los otros dos (ayuda adicional y ayuda de las herramientas) contiene toda la información comentada anteriormente.

LA CREACIÓN DE APLICACIONES

La forma más cómoda de trabajar con Watcom C++ es dentro del entorno Windows, sin duda alguna. Desde Windows se tiene acceso al Entorno Integrado de Desarrollo, a los archivos de ayuda para Windows y a versiones gráficas de muchas de las utilidades disponibles. Con toda seguridad la mayoría de programadores, sea cual sea la plataforma o plataformas para las que desarrolle, acabarán usando esta versión. A continuación se describe brevemente el procedimiento de trabajo más habitual para la creación de aplicaciones mediante el entorno Windows.

La herramienta de Watcom C++ más destacada para Windows es el EID. Se trata de una herramienta gráfica diseñada para centralizar las tareas más importantes relacionadas con la creación de nuevas aplicaciones. Si se utiliza el EID no es necesario aprenderse los parámetros de llamada de todas y cada una de las herramientas implicadas en la creación de un archivo ejecutable. La creación de los ficheros make, las llamadas a al preprocesador, compilador y enlazador se realizan de forma automática con los parámetros especificados en los menús, liberando al programador de las tareas más pesadas.

El trabajo con el EID se centra en el empleo de proyectos. Un proyecto es un fichero que contiene toda la información relativa a un programa que se esté desarrollando. Este fichero contiene una lista de los archivos que componen el programa y las instrucciones necesarias para construir cada elemento (por ejemplo, archivos ejecutables y librerías de enlace dinámico). A través del EID se pueden editar, compilar, ejecutar, depurar y modificar las aplicaciones.

Para construir una nueva aplicación lo primero que se debe hacer es crear un directorio en el que se almacenarán todos los archivos relacionados con la

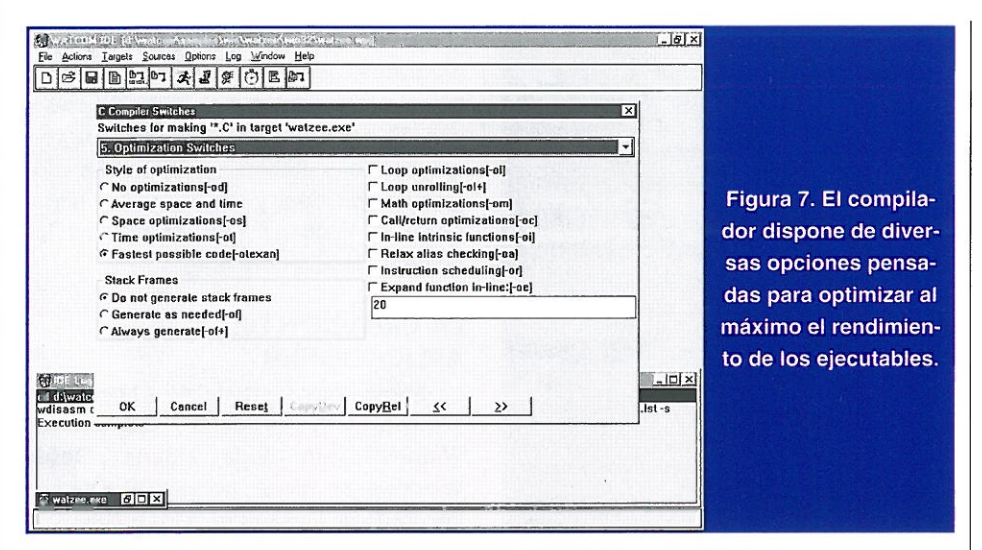


Figura 7. El compilador dispone de diversas opciones pensadas para optimizar al máximo el rendimiento de los ejecutables.

misma. Posteriormente se arranca el EID y se crea un nuevo archivo de proyecto en el directorio de la aplicación. Dicho archivo debe especificar al menos una plataforma de destino. Cuando se crea un proyecto, el EID muestra automáticamente un cuadro de diálogo con todas las opciones posibles. Si la aplicación va a ser ejecutada en otras plataformas se pueden añadir nuevos destinos posteriormente. En un principio dicho archivo estará vacío y se irá llenando de forma progresiva según se avance en el desarrollo. A continuación se irán creando los archivos fuente de C, C++ y ensamblador, además de otros archivos necesarios como iconos, gráficos y recursos. Cada uno de ellos se debe añadir a la ventana de la plataforma destino correspondiente con el fin de construir el ejecutable correspondiente. Esta ventana contiene una lista de todos los elementos y permite la visualización y edición de los mismos. El EID es inteligente en el sentido de que reconoce de forma automática el tipo de archivo y llama al editor más apropiado según sea el caso. Desde la ventana de plataforma también se pueden indicar opciones específicas para cada uno de los elementos, que serán tomadas en cuenta a la hora de construir la aplicación.

Una vez hecho todo esto ya se está en condiciones de crear los ejecutables y probar el programa. El siguiente paso es la corrección de los inevitables errores con el depurador de Watcom. Watcom Debugger es una potente herramienta de depuración con un interfaz gráfico mejorado que dispone de multitud de opciones. El aspecto más sobresaliente de Watcom Debugger es la posibilidad de volver hacia atrás en la ejecución. De este modo se deshacen todos los cambios producidos por la ejecución de una instrucción permitiendo así estudiar en detalle la ejecución del programa y las causas de los errores más difíciles de localizar.

Sea cual sea la plataforma para la que se desarrolle, se recomienda la utilización de la versión Windows

Por último es recomendable utilizar las herramientas de análisis de rendimiento *Watcom Execution Sampler* y *Watcom Execution Profiler*. Con ellas se pueden estudiar las partes del programa más usadas con el fin de mejorar la velocidad de ejecución del programa optimizando el código del mismo. La utilización de este tipo de herramientas incide directamente en la calidad final de las aplicaciones.

LA VERSIÓN 10.6

PowerSoft ha anunciado recientemente la versión 10.6 del compilador Watcom C++. La nueva versión no es más que una actualización de la 10.5 con algunas mejoras importantes centradas sobre todo en Windows 95, a la vez que se han subsanado errores encontrados en dicha versión.

La versión 10.6 incluye las MFC 3.2 para Windows 95 y Windows NT, y las MFC 2.52B para Windows 16 bits. Estas nuevas versiones de las clases de Microsoft incluyen soporte de TCP/IP mediante clases de Windows Sockets, soporte de MAPI para la creación de aplicaciones de correo y soporte de los controles comunes de Windows 95 y Windows NT 3.51. Otras mejoras introducidas son los nuevos tipos de iconos soportados por el editor de imágenes y la inclusión de todos los nuevos controles de Windows 95 en el editor de cuadros de diálogo.

PowerSoft también ha anunciado que todos los usuarios de la versión 10.5 tienen acceso a la actualización de Watcom C++ 10.6 de forma totalmente gratuita. Las páginas web de PowerSoft (http://www.powersoft.com) contienen toda la información referente a la nueva versión del compilador.

CONSIDERACIONES FINALES

Las empresas desarrolladoras de software se ven obligadas con mayor frecuencia de la deseada a adaptarse a nuevas plataformas para sus proyectos. Las razones de estos cambios son muchas, pero principalmente se deben a la aparición de nuevas plataformas de desarrollo y a la necesidad de adaptar programas existentes a otros entornos operativos. Para estas empresas constituye un gran esfuerzo económico y de tiempo el hecho de tener diferentes compiladores para cada una de las plataformas con las que trabaja. La solución ideal en estos casos es un compilador multiplataforma como Watcom C++, con características avanzadas, y con el que se puede crear todo tipo de programas, desde aplicaciones multimedia para Windows hasta juegos de gran calidad.

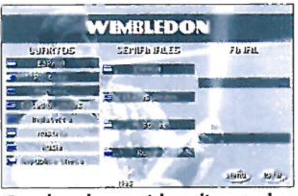
Watcom C++ ha demostrado ser una de las mejores herramientas de desarrollo disponibles en el mercado. Disfruta de amplio soporte técnico y de una excelente relación calidad/precio. Por muchas razones es una opción a tener muy en cuenta a la hora de adquirir un compilador de lenguaje C o C++.

ᴵIRTUAL TENIS

TODO EL TENIS MUNDIAL EN UN CD-ROM DE IMPACTO...



ːenos de juego creados en ːon nubes fractales.



Cuadro de partidos disputados en el torneo en curso.



ː torneos se pueden jugar ːompañía de un amigo.



Control de los jugadores con teclado o Joystick.

Virtual TENIS

POR SÓLO 2.995 PTAS. (IVA INCLUIDO)

RED · MODEM · CABLE SERIE · 4 JUGADORES

8 413042 303190 00001

CONTIENE: PC

REQUERIMIENTOS : PC 386 o superior, 4MB memoria RAM, lector de CD ROM, tarjeta VGA, disco duro, ratón recomendado. Compatible con Sound Blaster y Gravis Ultrasound.

CON LA GARANTIA DE: DDM DIGITAL DREAMS MULTIMEDIA

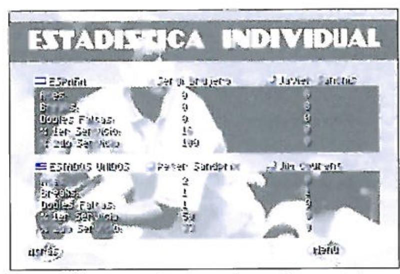
ABETO



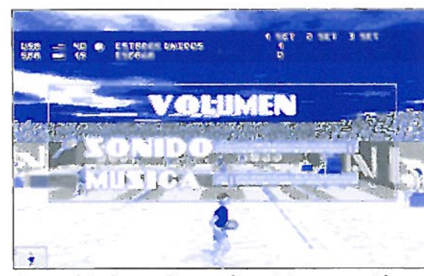
ːibilidad de jugar el Grand Slam completo: permite grabar ːeos y temporadas a la mitad.



ːpueden disputar los cuatro torneos del Grand Slam.



Completas estadísticas de cada partido.



Control independiente de música y sonido.

- ːsibilidad de conectarse con distintos jugadores mediante red, ːdem y cable serie.
- ːstema de menúes totalmente intuitivo.
- ːuda interactiva durante el desarrollo del juego.
- ːcil instalación en el disco duro.
- Jukebox para escuchar las distintas melodías que contiene.
- Cuatro diferentes superficies.
- Cada uno de los jugadores posee sus propias características.
- Permite elegir el número de sets que se desee disputar.
- Posibilidad de cambiar incluso el detalle gráfico.

CON LA GARANTÍA DE DDM DIGITAL DREAMS MULTIMEDIA

ːlicita VIRTUAL TENIS enviando este cupón o llamando al teléfono (91) 661.42.11* de 9 a 14 y de 15:30 a 18:30, o por Fax: (91) 661.43.86

ːeo que me envíen: ☐ **VIRTUAL TENIS por 2995 ptas. + 250 ptas. de gastos de envío.**

ːmbre y apellidos...Domicilio...Población........................

ːvincia...................................C.P ....F. de nacimiento........................Profesión..

FORMA DE PAGO:

Talón a ABETO EDITORIAL ☐ Contra-reembolso Teléfono .... Firma ,

Giro Postal nº...de fecha.........................

Tarjeta de crédito VISA nº ☐☐☐☐ ☐☐☐☐ ☐☐☐☐ ☐☐☐☐

ːcha de caducidad de la tarjeta......................................Nombre del titular, si es distinto...............................

ABETO

Rellena este cupón y envíalo a: ABETO EDITORIAL C/ Aragoneses, 7. 28100 Pol. Ind. ALCOBENDAS (Madrid)

DISCOS DE EMERGENCIA

David Aparicio

Al experimentar con ciertos aspectos críticos de nuestro sistema operativo favorito, podemos llegar a bloquear el arranque del mismo. Si se dispone de un disco de emergencia, al estilo de las utilidades Norton para DOS,se podrá restablecer el estado normal de Linux. Este mes se verá cómo hacer uno a medida

Sobre todo en los primeros pasos de instalación de un nuevo sistema, nuestra falta de experiencia nos lleva con frecuencia a realizar operaciones arriesgadas que pueden comprometer el arranque de nuestro PC. Muchas veces, una desesperante pantalla en blanco, o un mensaje de pánico de Linux termina con la paciencia de cualquiera, normalmente a altas horas de la madrugada.

Para evitar sobresaltos innecesarios, una de las primeras tareas aconsejables suele ser disponer de un disco de arranque que permita dar un "golpe de timón" a la situación. Los lectores acostumbrados a Slackware, sabrán que a lo largo del proceso de "setup" se nos permite crear un disco de emergencia. Sin embargo, suele ser habitual que este disco no satisfaga totalmente nuestros intereses, y deseemos personalizarlo algo más a nuestro gusto.

Como ya se comentó el mes pasado, mezclaremos cuestiones "de usuario", es decir, la operativa a seguir para construir un disco de arranque para nosotros desde la línea de comando, y "de hacker", explorando un poco más el núcleo, para averiguar cómo funciona la carga de un disco ram a partir de la unidad de discos flexibles. El primer tema se tratará en este capítulo, mientras que el segundo lo presentaremos el mes que viene.

¿QUÉ SE NECESITA?

Idealmente, en el disco flexible pondremos:

• El sistema de arranque: Por supuesto, al introducir el disco y reiniciar el PC, en primera instancia necesitamos que se ejecute el núcleo de Linux, para empezar a cargar procesos en nuestra máquina. Usaremos LILO para efectuar este paso.

• El sistema de ficheros: Nuestro disco debe poder ser arrancable, pero también necesita tener una estructura concreta (ext2 o minix) que albergue ficheros adicionales. De poco nos sirve que arranque el núcleo si luego no tenemos programas que lo aprovechen.

• Una shell: Adicionalmente, debemos obtener una linea de comando desde la que operar en nuestro sistema. Aunque podemos tender a pensar en instalar alguna a la que estemos acostumbrados (bash o similar), podemos vernos limitados por el tamaño disponible en nuestro disco de emergencia. Es este artículo veremos una alternativa, con una mini-shell denominada "sash".

• Utilidades de disco: "mount/umount" y "fsck" es el mínimo impresicindible para poder recuperar un disco dañado. Deberemos poder montar y desmontar unidades, formatearlas, o pasarles un test de integridad.

• Comandos externos: La capacidad de copiar, mover, renombrar o borrar ficheros es también importante para nuestro disco de emergencia. Si usamos la shell "sash", estas habilidades están incorporadas en el propio ejecutable. Si preferimos una shell más grande, deberemos añadirlas a nuestro disco.

• Un editor: Casi imprescindible para poder modificar ficheros de configuración erróneos, normalmente se incluirá el "vi", aunque si el espacio de disco lo permite, podemos ser más ambiciosos e introducir alguna herramienta mayor, del tipo de "Midnight Commander", un clónico del Comandante Norton. Además, si no estamos acostumbrados al editor estándar de Unix, quizá algo al estilo de DOS (joe o jed) puede sernos útil.

• Comandos extras: Si realmente nos queda mucho espacio disponible, podemos incluir utilidades de "segunda

línea", es decir, útiles sólo si disponemos previamente de todos los comandos antes mencionados. Entre las mismas, podemos incluir la pareja "tar/gzip", con la que podremos acceder a documentación o programas normalmente almacenados en un CD-ROM.

Una vez que tenemos la lista de requerimientos, procederemos a discutir cómo integrar todo lo mencionado en un sólo disco. En primer lugar, debemos repasar un concepto previo: el de los ejecutables que se enlazan estática o dinámicamente.

Con el fin de ahorrar espacio en un sistema, aparece el concepto de "enlazado dinámico" que se refiere a la capacidad de un programa de almacenarse en el disco en dos partes, una de propósito general (denominada librería dinámica), y otra particular a cada ejecutable, que forma el programa en sí. Como la parte de propósito general es compartida por todos los ejecutables de nuestro sistema, existe una sóla copia de dicho componente. Se carga en memoria una vez, y es referenciada durante la ejecución por todos los programas que la precisen.

Esta capacidad es muy interesante a partir de un tamaño de sistema determinado, porque ahorra disco duro y RAM. Sin embargo, como dicha librería es muy general, incluye gran cantidad de código que no todos los programas utilizan. En el caso de nuestro disco de arranque, por ejemplo, toda la parte de manejo de redes no es tan importante, y se podría obviar.

En este caso, los programas contenidos en el disco deberían ser "enlazados estáticamente", es decir, que la parte común y específica de los programas se unen en un sólo fichero, en tiempo de compilación. El programa resultante es más grande que su versión "dinámica", pero más reducido que la suma de aquel mas la librería común de enlace dinámico.

```
# LILO configuration file
#
# El arranque se instala en el disco flexible
boot = /dev/fd0
# No esperar antes de cargar el nucleo
delay = 0
# restablecer VGA estandar (80x25)
vga = normal
# El disco se carga en RAM o no
#ramdisk = 1440
# Parametros de arranque: Nombre del nucleo
image = /vmlinuz
# El filesystem esta en el propio disco (o ramdisk)
  root = /dev/fd0
# Etiqueta de carga: "Loading rescue"
  label = rescue
```

Cuadro 1: El fichero lilo.conf para arrancar con un disco flexible.

En definitiva, si el número de ejecutables en nuestro sistema es grande (como la parte estática de la librería viene a ser de 80K y el tamaño de la dinámica totaliza unos 700 Kbytes, podremos considerar que la frontera será tener más de nueve ejecutables), la versión dinámica de los mismos empieza a ser interesante. Sin embargo, si tenemos en cuenta el espacio total de un disco flexible, podemos imaginar que no tendremos demasiados lujos con un disco de emergencia.

EL ARRANQUE

En primer lugar , necesitamos un núcleo que contenga los drivers necesarios para aprovechar nuestro sistema. Si tenemos CD-ROM, este es uno de los dispositivos cuyo soporte es recomendable, para poder extraer archivos que fuesen a utilizarse. Los poseedores de controladoras SCSI, también necesitarán discos a medida. Si, por ejemplo, se tiene un sistema puro IDE o bien SCSI, se puede ahorrar espacio suprimiendo los drivers que correspondan a las controladoras que no nos interesen.

Un disco de emergencia es casi imprescindible para evitar sobresaltos al realizar ciertas operaciones con Linux

En definitiva, la idea es tener un núcleo pequeño pero adaptado al hardware disponible. Aunque los núcleos precompilados en la serie Q contemplan un abanico muy general y su aprovechamiento es inmediato, son un poco grandes para un uso continuado. En cuanto se tenga linux instalado mínimamente, se aconseja al lector que compile un núcleo a medida, y lo instale tanto para el arranque normal como para nuestro disco de emergencia.

Si es necesario, se puede seguir el artículo que se escribió al efecto en meses pasados. Para aquellos que no dispongan del mismo, podemos recordar, un poco telegráficamente, los pasos que se siguen para compilar un núcleo:

```
(instaladas las series D y K)
cd /usr/src/linux
make config
make dep
make zImage
(resultado en arch/
i386/boot/zImage)
(incluir núcleo en LILO)
```

En este punto, puede ser útil apuntar que el núcleo tiene, por defecto, un teclado americano. Si no ponemos programas especiales que reconfiguren el teclado (gastando nuestro precioso espacio en disco), tendremos la molestia de tener que adivinar dónde están los símbolos necesarios en cada momento.

Sin embargo, con un pequeño truco tendremos un núcleo con teclado castellano integrado. Suponemos que tiene ya instalado el soporte de teclado internacional (paquete "loadkeys"), y que estamos listos para recompilar nuestro núcleo, teclee lo siguiente:

```
cd /usr/src/linux/drivers/char
cp /usr/lib/kbd/keytables/es.map .
loadkeys —mktable es.map >
defkeymap.c
```

El fichero que hemos creado, defkeymap.c, contiene el mapa de teclado castellano. Ahora se puede recompilar el núcleo normalmente, y ya está.

Una vez que disponemos del núcleo adecuado, necesitaremos tomar un disco flexible y formatearlo. El sistema de

ficheros a elegir suele ser ext2 o minix. El primero es algo ineficiente en dispositivos pequeños, como nuestro disquetera, pero tiene la ventaja de poder suprimir el driver de minix del núcleo que compilemos, y ahorrar un poco más de espacio. En los núcleos más modernos es posible utilizar un disco ms-dos para ser montado como raíz, lo que tiene la ventaja de poder modificar nuestro disco de emergencia desde el mismo DOS (con lo que es posible recuperar el desaguisado incluso después de que haya ocurrido). Desgraciadamente, LILO y DOS parecen no entenderse por el momento para arrancar con el núcleo de Linux, desventaja que se solucionará próximamente.

Resumiendo, para nuestro ejemplo usaremos minix, que es más adecuado para dispositivos de pequeño formato. En este caso, el comando a usar es:

```
fdformat /dev/fd0H1440
mkfs /dev/fd0 1440
```

El primer comando se usa en discos sin formatear, y sólo es necesario realizarlo una vez para toda la vida del disco flexible. Es un proceso lento, al menos para el impaciente autor de esta sección, que equivale al "formateo a bajo nivel" de un disco duro.

El segundo nos permite grabar los sectores del disco con el sistema minix (esto se elige por defecto dado que no se ha especificado el tipo con el parámetro -t). Se observa que en el primer comando hemos cuantificado el tamaño del dispositivo (de 1.44 Mb) por el nombre. Esto es necesario, puesto que de un disco en blanco (sin sectores) el sistema no puede adivinar su geometria.

Con nuestro disco ya formateado, podremos proceder a crear el sistema de arranque para el mismo. Necesitaremos copiar el núcleo (por ejemplo, /root/minucleo), y decir a LILO que prepare el sector de arranque necesario para que el dispositivo se active en su momento. El descriptor del mismo se observa en la tabla 1. Este fichero (por ejemplo /root/lilo.conf) se usa de la siguiente forma

```
mount /dev/fd0 /mnt
cp /root/minucleo /mnt
mkdir /mnt/etc
mkdir /mnt/boot
cp /boot/boot.b /mnt/boot
cp /root/lilo.conf /mnt/etc
lilo -r /mnt
umount /mnt
```

Para aprovechar el espacio y minimizar los ficheros requeridos, todos los programas están enlazados estáticamente

Con estos comandos hemos grabado el núcleo que hayamos escogido en el disco, y lo hemos preparado para que arranque con éste. El mínimo de ficheros que espera el núcleo de linux para arrancar es:

```
/dev/tty1
/etc/init
```

Si existen ambos ficheros en nuestro disco flexible, el proceso "init" es invocado por el núcleo, siendo capaz de tomar el control del sistema. Normalmente este ejecutable es un programa complejo que requiere varios ficheros de configuración, (/etc/inittab y /etc/rc.d/*) y ejecutables (/sbin/getty, /bin/login) que no resultan muy interesantes si símplemente deseamos tener una línea de comando con la que controlar el sistema. Para nuestro disco de emergencia, bastará con que el ejecutable "init"sea una simple shell renombrada.

```
mount /dev/fd0 /mnt
mkdir /mnt/bin
mkdir /mnt/dev
mkdir /mnt/mnt
mkdir /mnt/tmp
cp /mipath/sash /mnt/bin
mknod /mnt/dev/tty1 c 4 1
cd /mnt/etc
ln -s ../bin/sash init
cd ../bin
ln -s sash sh
cd
umount /mnt
```

Ahora nuestro disco ya es capaz de arrancar y además proporcionarnos una shell con una serie mínima de comandos. Podremos incluir los programas adicionales que se deseen para tener operativo el sistema, los cuales, recordando la discusión de antes, deben haber sido compilados estáticamente. Supuesto un programa denominado 'ejem', el comando que produciría una versión estática del mismo sería de la siguiente forma

```
cc -static -O -o ejem ejem.c
```

Lo cual contrasta con una compilación de ejecutable dinámico, que podría ser así

```
cc -o ejem ejem.c
```

Esto puede complicarse algo si nos estamos enfrentando a algún programa de tamaño respetable, donde se utiliza un fichero denominado Makefile para realizar la compilación, y que contiene los comandos y reglas que conduzcan al éxito de la misma. En ese caso, y aunque no sea una "receta" segura, la variable LDFLAGS suele ser la que debería contener la directiva que nos interesa, -static. Como por defecto todas las utilidades están pensadas para ser compila-

Una vez que arrancando el sistema, no se accede más al disco flexible, permitiendo trabajar de forma cómoda

das dinámicamente, deberemos observar detenidamente cada caso.

El total de programas que se han incluido en nuestro disco de arranque de ejemplo han sido:

- sash

Esta cantidad "sorprendente" de ejecutables se debe a que todos los comandos que hemos comentado anteriormente (manejo de archivos, compresión, montado, chequeo de integridad, listado, creación y gestión de directorios y ficheros, edición de los mismos) se encuentran incluidos en la propia shell, y se pueden observar en el listado de la tabla 2.

Finalmente, para poder acceder al disco duro, necesitamos también crear los dispositivos de acceso a los discos. En el caso de discos IDE:

- /dev/hda*
- /dev/hdb*
- /dev/hdc*
- /dev/hdd*

Y para discos SCSI

- /dev/sda*
- /dev/sdb*
- /dev/sdc*
- /dev/sdd*
- /dev/sde*
- /dev/sdf*
- /dev/sdg*

Sin olvidar que como mínimo necesitaremos /dev/tty1 como pantalla, y probablemente se deberá añadir los dispositivos de CD-ROM, unidades flexibles de discos, /dev/null, y otros. Podemos ahorrar tiempo y molestias utilizando el script /dev/MAKEDEV, que realiza automáticamente la creación de todos los dispositivos mediante el comando "mknod". Desde aquí se recomienda estro último, con los siguientes comandos de ejemplo. Sólo es necesario:

```
cd /dev
MAKEDEV std
MAKEDEV console
MAKEDEV tty1
```

Para crear dispositivos de almacenamiento opcionales deberemos utilizar como parámetro el dispositivo requerido. Por ejemplo:

```
MAKEDEV sda
MAKEDEV hda
MAKEDEV fd0
```

Para varios tipos de CD-ROM, podemos usar aquel que convenga para cada caso. El último corresponde a unidades SCSI.

```
MAKEDEV idecd
MAKEDEV sbpcd
MAKEDEV sonycd
MAKEDEV scd
```

Por último, se pueden crear todos los dispositivos con un sólo comando, que puede tardar un poco en ejecutarse y es:

```
MAKEDEV generic
```

Adicionalmente, necesitaremos directorios adicionales, como algún punto de montaje para acceder a las unidades. El directorio clásico suele ser /mnt para nuestros discos duros, pero cada usuario puede crear tantos como desee. En nuestro ejemplo, podemos acceder a la tercera partición del primer disco IDE (con formato ext2), una vez que nuestro disco de emergencia esté arrancado, con el comando:

```
mount -t ext2 /dev/hda3 /mnt
```

También es importante el directorio /proc, que permitirá montar el sistema de ficheros de diálogo con el núcleo, y sin el que no funcionan comandos como 'ps'.

```
xterm
david:/dosa# sash
[SA]SH v1.01a (c) 1993 by David I. Bell, modified 1996 by David Aparicio

[/dosa] find
[external]
.
./vmlinuz
./dev
./dev/tty1
./dev/fd0
./dev/sda3
./bin
./bin/sh
./bin/sash
./etc
./etc/init
./etc/lilo.conf
./etc/termcap
./proc
./mnt
./boot
./boot/boot.b
./boot/boot.0200
./boot/map
./tmp
[/dosa]
```

Figura 1: La shell de emergencia SASH

Con todo esto, y algún que otro detalle de menor importancia, ya habremos completado nuestro disco salvador. En el ejemplo que se adjunta con la revista, hemos ocupado la mitad del total del disco de 1.44 Mb, como se observa en la figura 1. Cada lector puede ajustar y experimentar diferentes configuraciones, para aprovechar lo mejor posible el espacio disponible.

EL DISCO RAM

Llegado este punto y tras unos cuantos ensayos con nuestro nuevo disco, podremos observar fácilmente que, tras el arranque, nuestro disquete está frecuentemente ocupado por accesos de los programas ejecutados desde el mismo. Esto es molesto cuando estamos trabajando puesto que la unidad tiene un comportamiento lento en comparación con un disco rígido, y además impide que podamos introducir otros discos en nuestra unidad. Es por ello que casi nunca se usa un disco de arranque de esta forma, sino que el núcleo copia durante su arranque una imagen del disco en RAM y se acede desde allí.

Una vez que arrancado el sistema, el disco flexible ya no se accede más, permitiendo trabajar de forma cómoda. Adicionalmente, como el sistema de ficheros que vemos es una copia en memoria del contenido que había en el disco, podemos crear y borrar ficheros del mismo sin que el original se vea afectado, con lo que disminuimos la posibilidad de dañar nuestro disco de arranque con una operación inadecuada (inhabilitar el único disco de emergencia disponible en medio de una reparación es una experiencia bastante

alias [name [command]]	-mkdir dirname ...
cd [dirname]	-mknod filename type major minor
-chgrp gid filename ...	-more filename ...
-chmod mode filename ...	+mount [[-t type] devname dirname]
-chown uid filename ...	-mv srcname ... destname
-cmp filename1 filename2	+net interface ipaddr
-cp srcname ... destname	-printenv [name]
-dd if=name of=name [bs=n] [count=n] [skip=n] [seek=n]	+ps
-echo [args] ...	-rm filename ...
-ed [filename]	-rmdir dirname ...
+e2fsck devname	setenv name value
exec filename [args]	source filename
-grep [-in] word filename ...	swapon devname
+gunzip filename	swapoff devname
+gzip filename	-sync
+halt [-r]	-tar [xltlv]f devname filename ...
help	-touch filename ...
-kill [-sig] pid ...	umask [mask]
-ln [-s] srcname ... destname	+umount filename
-ls [-lid] filename ...	unalias name
	vi [filename ...]

Cuadro 2: Comandos disponibles en SASH.

desagradable por la que no se recomienda pasar). En la tabla 1, se observa una línea (ramdisk) que debe ser activada (eliminando el primer carácter de la línea, que la marca como un comentario) para aprovechar la habilidad que se ha comentado.

Junto con este artículo, se presenta un ejemplo de disco de emergencia, que podremos utilizar partiendo de un disco formateado de 1.44 Mb. Desde Linux, la secuencia de comandos es, supuesto que el fichero rescue.gz está en el path 'mi_path':

```
cd /mi_path
gzip -d rescue.gz
dd if=rescue of=/dev/fd0
```

Desde DOS, deberemos disponer de un CD-ROM de Slackware, donde se encuentran las utiilidades GZIP y RAWRITE, y repetiremos la misma secuencia que efectuamos para crear los discos de boot y root:

```
CD \MI_PATH
GZIP -d RESCUE.GZ
RAWRITE RESCUE A:
```

Si se desea examinar o modificar el disco, podremos efectuar los siguientes pasos:

```
(introducir el disco 'rescue')
mount /dev/fd0 /mnt
cd /mnt
(hacer operaciones sobre /mnt)
cd
umount /mnt
```

LA SHELL SASH

El haber incluido todos los programas necesarios para administrar mínimamente el sistema en un solo ejecutable, nos permite ocupar un total de 800 Kb en el disco de emergencia. Este espacio tan pequeño se debe a una shell muy especial, cuyo autor es David I. Bell, y que ha sido modificada por el autor de esta columna para incluir algunas utilidades adicionales ("vi", "ps", "ifconfig/route", "swapon/swapoff", "gzip" y "fsck"). Hemos de comentar que debido a tal modificación, este programa está en betatesting, y es posible que no funcione con alguna configuración. Todos los errores se asumirán como achacables a la modificación que se ha hecho, y no al autor del programa original.

A partir del listado de la tabla 2, pasaremos a comentar brevemente todos los comandos que se ofrecen:

- Un alias es un seudónimo que nos permite abreviar o renombrar otros comandos. Su uso es muy sencillo. Con el comando "alias" se puede especificar un número variable de parámetros. El primero indica la etiqueta a asociar, y el resto de la línea el comando representado por ésta. Dicha asociación se puede destruir con el comando "unalias". Por ejemplo, para tener el comando 'cat', basta teclear:

  alias cat more

- Los comandos "cd" y "mkdir" permiten cambiar de directorio o crear uno nuevo, respectivamente. "ls" visualiza el contenido de un directorio, y "rmdir" borra éste. "cp" y "mv" copian o mueven ficheros en el sistema. Si el último parámentro es un directorio, podemos copiar múltiples ficheros en origen con un sólo comando. "rm" borra ficheros del sistema.
- "chown", "chgrp" y "chmod" son los comandos que permiten cambiar los permisos y propietarios de todos los ficheros del sistema. "umask" establece unos permisos por defecto, que se usarán cuando creemos nuevos ficheros. No los describimos aquí, igual que los del punto anterior puesto que se encuentran perfectamente detallados en el man del sistema, y son muy corrientes.
- "cmp" compara los contenidos de exáctamente dos ficheros indicando si hay diferencias o no.
- "dd" permite copiar fragmentos de un fichero en otro, controlando el tamaño de bloque usado en la transferencia (bs), el número de bloques a copiar (count), el número de bloques a saltar en origen (skip), el de bloques a saltar en destino (seek), el nombre del fichero origen (if) y el de destino (of). Sólo estos dos últimos parámetros son obligatorios. Tambien podemos leer o escribir sectores de disco, lo cual es muy útil (si sabemos lo que estamos haciendo). Por ejemplo, para copiar 20Kbytes (en bloques de Kbyte), del fichero 'a' al 'b', a partir de 2 Kbytes de destino, teclearíamos:

  dd if=a of=b bs=1k count=20 seek=2

- "e2fsck" es el comando de comprobación de cosnistencia para unidades de tipo "ext2", que son las estándar en Linux. Basta pasar como parámetro la unidad a chequear, que debe estar desmontada en ese momento. De hecho, la unidad 'raíz' de nuestro disco duro sólo se puede chequear manualmente con un disco de arranque (la raíz del sistema de ficheros no se puede desmontar, al menos en la versdión 1.2.13).
- "grep" permite buscar una cadena de caracteres, normalmente en varios ficheros simultáneamente.
- La pareja "gzip" y "gunzip" comprimen y descomprimen ficheros con la

extensión "gz" o "tgz". La última indica que el fichero está, además, en formato 'tar', el cual se explica más adelante.

• "sync" y "halt" se usan para sincronizar los discos montados (grabar los bloque modificados del caché de memoria al disco) y para parar el sistema. Este último, con el comando '-r', rearranca tras parar. No es aconsejable rearrancar el sistema sin desmontar los discos, con el comando que se explica más adelante.

• "ps" y "kill" permiten visualizar y cancelar procesos que estén corriendo en la máquina. Su funcionamiento es similar a los comandos indicados en el manual on-line.

• "ln" permite crear links entre ficheros. Un link es una entrada en la estructura de directorios que no tiene contenido propio, sino que comparte el de otro fichero normal del sistema. Esto permite acceder a un mismo contenido con varios nombres diferentes. "mknod" crea ficheros especiales, que representan a los dispositivos del sistema. Cuando leemos o escribimos en este fichero, estamos obteniendo o grabando datos del dispositivo asociado.

• "more" muestra el contenido de un fichero (supuestamente, de texto) por pantalla, haciendo una pausa tras cada pantalla. En esta pequeña shell, este comando hace las veces de 'cat', cuya diferencia estriba en que no se hace esta pausa.

• "net" es un comando usado en redes, para dar una dirección IP a una tarjeta de red. Es una combinación muy simplificada de las utilidades "ifconfig" y "route", y se usa para activar las características de comunicaciones IP en el núcleo.

• "printenv" y "setenv" pueden mostrar e insertar variables de entorno, para recordar paths o cualquier utilidad que se nos ocurra.

• "mount" y "umount" se utilizan para acceder al sistema de ficheros de una unidad concreta, a partir de un directorio determinado de nuestro árbol actual. Se debe especificar el tipo de sistema de la unidad que se accede, y por defecto es "minix". El comando de desmontado se debe usar antes de rearrancar, apra asegurar que la información se graba correctamente en la unidad.

• "swapon" y "swapoff" activa y desactiva el swap (memoria virtual) en una unidad dedicada a tal efecto, con lo que la memoria total del sistema varía y se pueden ejecutar programas más grandes de los que normalmente permitiría la memoria física instalada.

Los componentes principales del disco son LILO para el arranque, un núcleo y una shell

• "touch" permite cambiar el tiempo de acceso de un fichero (o crear uno que no existía) como si hubiese sido modificado con un editor.

• "tar" es un comando que permite visualizar (comando 't') o desempaquetar ('x') una estructura de ficheros y directorios a partir de un archivo original. La distribución SlackWare utiliza una combinación de éste comando y del compresor "gzip" para almacenar las utilidades que la componen. Mediante éste, podemos instalar o actualizar paquetes a partir de un CD o disco con los ficheros originales.

• "vi" por último, es el editor estándar en Unix y es muy útil para modificar ficheros de configuración del sistema. Aquellos usuarios que no quieran utilizar este editor, dado su carácter tan particular, pueden usar uno externo (jed, joe u otro similar) que se asemeja más a los que se suelen encontrar en DOS.

Con este somero repaso a los comandos disponibles (que se pueden ver dentro de la propia shell, con el comando "help") podemos observar la variedad de utlidades que se han intentado incluir en este programa, y que debe permitirnos operar con un mínimo de libertad para recuperar un posible sistema dañado.

Finalmente, el fichero ".aliasrc", si se encuentra en el HOME del usuario que invoca esta shell (el directorio raíz del disco flexible en el caso del disco de emergencia), permite la ejecución automática de los comandos que contenga, como si el usuario los teclease manualmente en el arranque. Un ejemplo de contenido de este fichero podría ser:

```
alias cat more
alias ls ls -l
echo Disco de emergencia de SoloP
```

La primera línea hace que el comando "more" se pueda ejecutar al teclear "cat". La segunda incorpora el comando "-l" cada vez que tecleamos "ls" (formato largo al listar contenidos de directorios). La última línea muestra un mensaje de bienvenida cuando se invoca la shell.

CONCLUSIONES

En este mes hemos visto la utilidad de tener un disco de arranque, por si hace falta restablecer el buen funcionamiento de nuestro sistema. Dado que el espacio disponible es más bien escaso, debemos tratar de incluir, además del núcleo, tantas utilidades como sea posible, pero siempre con buen juicio, para tener algo de comodidad a la hora de usar este disco. Los ejecutables tradicionales, dinámicos, tienen una menor utilidad, debido a la necesidad de incluir las librerías y el sistema de enlazado dinámico en el mismo disco, lo que se lleva casi dos terceras partes del mismo.

Además, el tener excesivo número de ejecutables estáticos también redunda en un gasto importante de espacio, puesto que tendremos repetido en el disco la parte del código común de las librerías que se hayan enlazado. En resumen, que debemos tener ejecutables estáticos, y el menor número de ellos posible, todo dentro de un margen razonable. La solución propuesta aquí se basa en una shell que incorpora todos los ejecutables que se usan normalmente, como único ejecutable del sistema de ficheros. Como anécdota, este solo programa ocupa menor tamaño que la librería dinámica de linux completa.

Los usuarios que lo deseen, podrán incorporar otros ficheros a este mínimo marcado aquí, para obtener un entorno confortable, que dicho sea de paso, deseamos desde aquí a los lectores que no haya que usar con demasiada frecuencia. Hasta otra.

SEGMENTACIÓN E INTERCONEXIÓN DE REDES (I)

María Jesús Recio

En el número anterior se describió el modelo teórico de una red de área local, sus principales características y elementos; se habló de diferentes topologías, y métodos de acceso al medio. Se explicó la diferencia existente entre la topología física que se implementaba y la topología lógica con la que la red estaba funcionando realmente. Además, se describieron los principales modelos de redes de área local establecidos, así como sus variaciones.

Una vez que se tienen definidas las redes de área local, se va contar cómo pueden las redes ampliarse, segmentarse e interconectarse unas con otras.

La interconexión de redes se puede establecer a varios niveles: desde el nivel físico, a través de un dispositivo llamado hub (concentrador) hasta niveles más altos (niveles del modelo OSI de la ISO, se entiende) a través de dispositivos como son un (puente) bridge, o router (encaminador).

Para la segmentación de redes, y teniendo en cuenta uno de los motivos por el que se realiza esta operación, mejorar el rendimiento de la red, es necesario emplear dispositivos inteligentes, como pueden ser un encaminador o un puente.

En este artículo se estudiaran detenidamente algunos de los dispositivos interconectantes; el resto se verán en el próximo artículo.

¿POR QUÉ SE INTERCONECTAN REDES?

Hace algunos años era impredecible la evolución que las comunicaciones, en el mundo de la informática iba a tener: no se podía prever que fuese necesaria la interconexión ya no sólo de varios ordenadores sino de cientos de ellos. No basta con tener los ordenadores de una sala conectados, es necesario conectarlos a su vez con los ordenadores del resto de las salas de una empresa, con el resto de las sucursales de una empresa situadas en distintos puntos geográficos.

La interconexión de redes permite, si se puede decir así, ampliar el tamaño de una red. Sin embargo el término interconexión se utiliza para unir redes independientes, no para ampliar el tamaño de una.

El número de ordenadores que componen una red es limitado, depende de la topología elegida, (recuérdese que en la topología se define el cable a utilizar) aunque si lo único que se quisiera fuera sobrepasar el número de ordenadores conectados, podría pensarse en simplemente segmentar la red. Sin embargo existen otros factores a tener en cuenta.

Cuando se elige la topología que va a tener una red se tienen en cuenta factores, como son la densidad de tráfico que ésta debe soportar de manera habitual, el tipo de aplicaciones que van a instalarse sobre ella, la forma de trabajo que debe gestionar, etc; ésto debe hacer pensar en que, uno de los motivos por el que se crean diferentes topologías es, por tanto, el uso que se le va a dar a la red. De aquí se puede deducir que, en una misma empresa puede hacerse necesaria no la instalación de una única red, aunque sea segmentada, sino la implantación de redes independientes, con topologías diferentes e incluso arquitecturas diferentes y que estén interconectadas.

Inicialmente los ordenadores se instalaban de forma aislada, después aparecieron las redes locales que interconectaban varios ordenadores; hoy en día se hace tan nesesaria la expansión de redes como la interconexión de las mismas

NECESIDAD	SOLUCIÓN
Debido a la necesidad de manejo de aplicaciones que producen un trasiego importante de información aumenta el tráfico en la red; esto lleva a que baje el rendimiento de la misma.	Dividir la red actual en varios segmentos: segmentar la red.
	Crear un nuevo segmento de red en el que se pondrán los nuevos puestos, que por disposición física pueda ser conveniente que pertenexcan al nuevo segmento creado en la misma.
Se tiene que ampliar el número de puestos que forman la red, pero se necesita mantener el rendimiento de la red.	
Se tiene la necesidad de unir dos redes exactamente iguales en la empresa.	Se puede optar por definir una de ellas como un segmento de la otra y unirla de esta forma; o bien, interconectar las dos redes con un dispositivo de nivel bajo.
Se tiene la necesidad de unir dos o más redes con diferentes topologías pero trabajando con los mismos protocolos de comunicaciones.	Es necesario la interconexión de ambas redes a través de dispositivos interconectantes de nivel medio.
Se tiene la necesidad de unir dos o más redes totalmente diferentes, es decir, de arquitecturas diferentes.	Es necesario la interconexión de ambas redes a través de dispositivos interconectantes de nivel alto.

Tabla 1.: Segmentar e interconectar.

Habitualmente la selección del tipo y los elementos físicos de una red, se ajusta a las necesidades que se tiene; por este motivo pueden encontrarse dentro de un mismo edificio, varias redes con diferentes topologías, y con el tiempo puede surgir la necesidad de interconectarlas.

Se puede ver que por diferentes razones se hace necesaria tanto la segmentación como la interconexión de redes, y que ambos conceptos a pesar de llevar a un punto en común, parten de necesidades distintas.

La tabla 1 refleja de forma escueta diferentes casos en los que se plantea la necesidad de segmentar y/o interconectar redes, dando la opción más idónea para cada uno de los casos planteados.

Las figuras 1.a, 1.b y 1.c reflejan una red inicial y dos implementaciones posibles para la expansión de la misma.

SEGMENTACIÓN: SUS NECESIDADES

Segmentar una red consiste en dividirla en subredes para así poder aumentar el número de ordenadores conectados a ella y/o el rendimiento de la misma.

Cuando se segmenta una red, lo que se está haciendo es crear subredes pequeñas que, por decirlo de alguna manera, se autogestionan, de forma que la comunicación entre segmentos solo se realiza cuando es necesario, es decir, cuando un nodo de un segmento quiere comunicarse con un nodo del otro segmento; mientras tanto cada segmento de la red está trabajando de forma independiente por lo que en una misma red se están produciendo varias comunicaciones de forma simultánea; evidentemente esto mejora el rendimiento de la red.

La tabla 2 refleja las longitudes máximas de los segmentos dependiendo de las diferentes topologías de red.

Llegado este punto sería bueno poner un ejemplo recordando el funcionamiento del método de acceso al medio CSMA/CD , utilizado por redes 802.3 como ejemplo práctico de la necesidad de segmentación de una red. En este método, cuando una estación quería transmitir, escuchaba el medio y cuando lo detectaba vacío (no ocupado por otra estación), entonces empezaba a transmitir; no obstante, la estación se quedaba escuchando por si durante la transmisión se produjeran colisiones; llegado este caso, se esperaría un tiempo aleatorio para volver a intentarlo. El método es bueno cuando el tráfico de la red es bajo, ya que no se producen demasiadas colisiones; sin embargo, cuando el tráfico en la red aumenta, debido básicamente a que muchas estaciones quieran transmitir de forma simultánea, aumenta el número de colisiones, y por tanto se incrementa directamente el número de retransmisiones lo cual repercute en que el rendimiento de la red baje de manera importante. Al segmentar la red, el tráfico de ésta se ve reducido, pues se crean dos subredes independientes.

Ambas subredes sólo van a tener que "sincronizarse" cuando un nodo de una de ellas quiera comunicarse con un nodo de la otra; en este caso el dis-

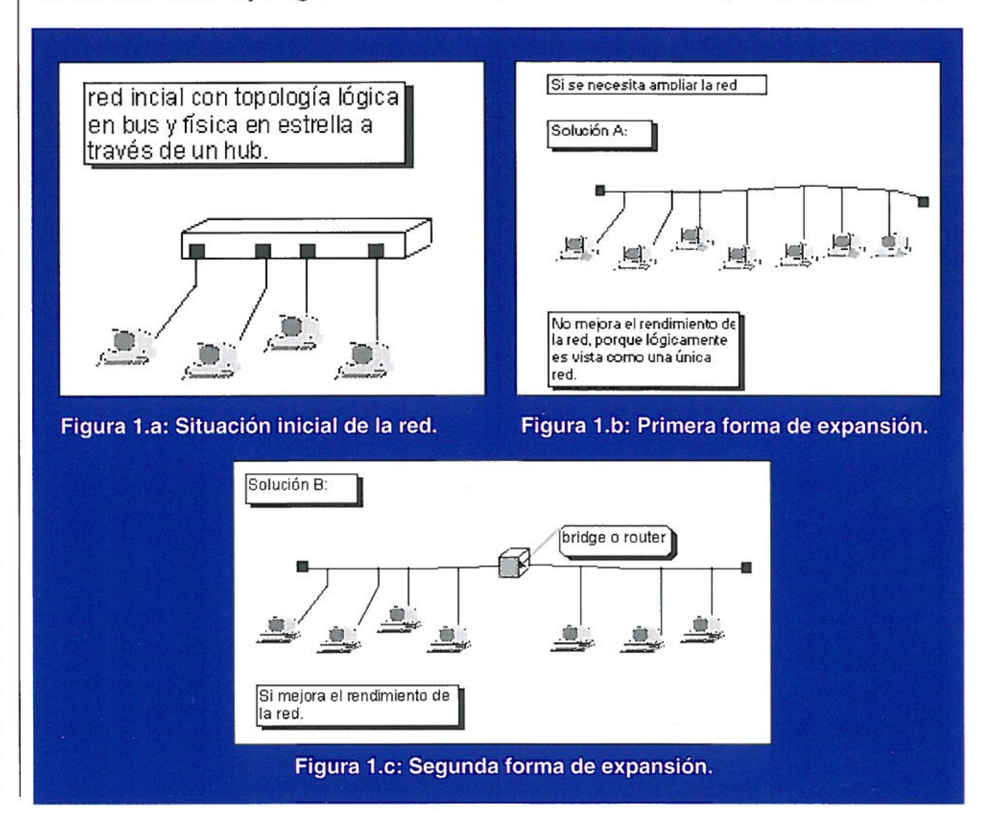


Figura 1.a: Situación inicial de la red.

Figura 1.b: Primera forma de expansión.

Figura 1.c: Segunda forma de expansión.

TOPOLOGÍAS	LONGITUD MÁXIMA DEL SEGMENTO
Ethernet gruesa	500 metros
Ethernet fina	185 metros
Ethernet de par trenzado	100 metros
Ethernet de fibra óptica	2.000 metros
Token-Ring de par trenzado	100 metros

Tabla 2.: Longitud del cable en segmentos de diferentes topologías.

positivo que segmenta la red se encargará de realizar dicha operación. La figura 2 refleja esta situación.

El dispositivo que se utiliza para segmentar una red debe ser inteligente, ya que debe ser capaz de decidir hacia qué segmento debe enviar la información llegado a él: si hacia el mismo segmento desde el que la recibió, o hacia otro segmento diferente.

Abstrayéndose de algunos detalles, es fácil pensar que segmentar una red, ya que se habla de subredes, es como interconectar redes diferentes. Sin embargo, cuando se habla de segmentar se hace referencia a una única red; esto lleva asociado lo siguiente: una única topología, una única pila de protocolos de comunicaciones,y un único entorno de trabajo. Cuando se habla de interconectar redes, en la mayoría de los casos, las redes tienen, como mínimo topologías diferentes. No obstante, si debe destacarse que los dispositivos que se utilizan para segmentar redes coinciden con algunos de los dispositivos que son usados para interconectar redes diferentes.

Dependiendo del tipo de protocolos que se utilicen en la red segmentada , así como del dispositivo que se utilice para realizar esta segmentación puede hacerse necesario o no el atribuir a cada segmento una dirección de red diferente. Por ejemplo en redes trabajando con Netware de Novell (y protocolos IPX/SPX) si se hace necesario el dar a cada segmento una dirección de red diferente; sin embargo para redes que trabajen con la familia de protocolos TCP/IP, no es necesario; basta con que cada estación tenga su propia dirección IP, y que no aparezcan dos estaciones con la misma dirección, independientemente de si están o no en el mismo segmento de la red.

Existen diferentes motivos por los que se puede hacer necesario la segmentación de una red, como pueden ser:

- Necesidad de sobrepasar el número de nodos que la topología permite.
- Mejorar el rendimiento de una red en la que ha aumentado el tráfico.
- Necesidad de sobrepasar el número de nodos que la topología permite.

La limitación del número de nodos en una red de área local viene impuesta por varios factores, como son el método de acceso al medio que se utiliza, el tipo de cable, el ancho de banda empleado para la transmisión, etc.

Llegado este caso sería necesaria la reestructuración de las conexiones; obviamente si además de ampliar la red se piensa en mejorar el rendimiento de ésta, no sirve de mucho conectar una única estación a un segmento, ya que todas las comunicaciones de esta van a ir dirigidas hacia el otro segmento de la red, por lo que el tráfico de un segmento y otro estaría bastante desproporcionado. En este caso lo que se debería hacer, siempre que la disposición física de las estaciones lo permitiera, es equilibrar los dos segmentos, moviendo estratégicamente estaciones del segmento inicial hacia el que estamos creando. La complejidad de esta operación varía muchísimo dependiendo de la topología física que implementa la red: si lo que se tiene es una topología en estrella a través de un concentrador, es relativamente sencillo cambiar la conexión de una estación de un concentrador a otro, sin embargo si la topología es un anillo físico, este cambio se presenta bastante más complicado que el anterior.

Además, hay que tener en cuenta el sistema operativo que maneja la red; si sobre la red se esta ejecutando Netware de Novell, con un único servidor de ficheros, evidentemente las comunicaciones serán desde el servidor hacia el resto de las estaciones de la red, por lo que en este caso es difícil hacer una división de la red en segmentos.

MEJORAR EL RENDIMIENTO DE UNA RED EN LA QUE HA AUMENTADO EL TRÁFICO

En ocasiones, una red que inicialmente funcionaba bien, con un tiempo de respuesta aceptable, empieza a perder prestaciones; el motivo es claro: de forma paulatina se ha ido incrementado el número de comunicaciones que la red debe gestionar, por motivos como:

- Los usuarios empiezan a aprovechar la conexión en red para compartir más recursos.
- Las aplicaciones que se han ido instalando (como las aplicaciones multimedia) requieren una densidad de transmisión mucho más grande; no es lo mismo enviar un fichero de texto de 100 K, que transferir imágenes y/o sonido.

Debido a todo esto, y aunque el número de estaciones que forman la red esté por debajo del límite que la topología aconseja, el rendimiento de la red baja considerablemente y empieza a ser lamentoso trabajar en estas condiciones.

Existen diferentes formas de paliar este problema: una de ellas, la más drástica, es cambiar algún elemento físico de la red: por ejemplo, sustituir el cable que implementa la red por uno que pueda soportar velocidades mayores, cambiar las tarjetas de red por

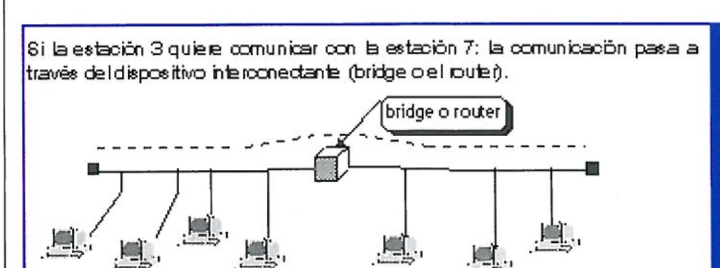


Figura 2.: Forma de comunicarse segmentos de una red.

otras más rápidas, e incluso cambiar la topología empleada. Una solución menos concluyente consiste en segmentar la red. Dividirla estratégicamente en dos subredes, reduciendo de esta forma el tráfico en cada una de ellas. Por ejemplo, sobre una red inicial repartida por varias aulas de un centro, se pueden crear subredes por aula, de forma que en cada aula se mejorará el rendimiento de la red.

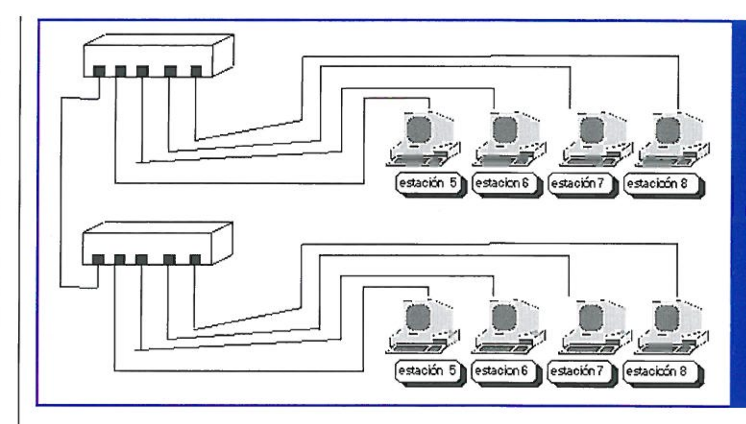


Figura 3.: Configuración de una red en bus lógico a través de la unión de dos hubs.

DISPOSITIVOS QUE PERMITEN INTERCONECTAR Y SEGMENTAR REDES

Los principales elementos que nos permiten interconectar y/o segmentar redes son: concentradores (hubs), puentes, (bridges), encaminadores (routers) y pasarelas (gateways). Cada uno de estos dispositivos trabaja a un nivel diferente de la pila de niveles del modelo de interconexión de sistemas abiertos (OSI), por tanto cada uno de ellos está capacitado para realizar tareas de los niveles en los que trabaja. Mientras más alto sea el nivel al que un dispositivo trabaja más diferencias pueden existir entre las redes que interconecta: un dispositivo que trabaje a nivel físico únicamente podrá interconectar redes iguales, con la misma topología y pilas de protocolos, mientras que un dispositivo que trabaje a nivel de aplicación será apto para interconectar redes muy distintas. En la figura se puede observar gráficamente el nivel en el que trabajan cada uno de ellos.

El elemento elegido para interconectar dos redes no se selecciona únicamente valorando las diferencias entre las redes a interconectar; existen otros factores, como son la velocidad de transmisión (mientras más inteligente sea un dispositivo más tiempo de retardo introduce en la transmisión de la información llegada hasta él, ya que debe procesarla), la capacidad de encaminamiento, etc. que son tenidos en cuenta a la hora de optar por un dispositivo u otro.

Existe un dispositivo más, que aunque estrictamente no se pueda decir que permite interconectar/segmentar una red, esta muy relacionado con este mundo y por tanto es necesario nombrar: el repetidor.

La figura 4 esquematiza los niveles a los que trabajan los diferentes dispositivos mencionados en el párrafo anterior.

REPETIDOR

Se trata del dispositivo más sencillo; en realidad como ya se ha dicho anteriormente, no permite interconectar ni segmentar redes, se emplea únicamente para amplificar la señal transmitida.

A veces, debido a diferentes causas como pueden ser, el medio que rodea los cables de una red (la cantidad de ruidos e interferencias que se produzcan alrededor), el tipo de cable empleado, la distancia, etc, la señal transmitida se distorsiona de forma que cuando llega a su destino, en el mejor de los casos, este, no la interpreta correctamente, por lo que se producen errores de transmisión y de nuevo, el rendimiento de la red disminuye. En estos casos es necesario utilizar un repetidor de señal, el cual no es más que un dispositivo que recibiendo una señal atenuada, la reenvía regenerándola previamente; de esta forma se pueden solventar los problemas reflejados anteriormente.

Por tanto, se puede decir, que un repetidor no es sino un dispositivo eléctrico, que no entiende de redes, direcciones, encaminamiento, sólo de señales eléctricas que recibe y reenvía, es decir, lee pulsos distorsionados del cable y los envía regenerándolos previamente. A modo de resumen, podemos decir que las principales características de un repetidor son:

- Regenera las señales para que puedan viajar más lejos.
- Se usa básicamente en redes con topología en bus, como Ethernet.
- Opera al nivel más bajo de la pila de protocolos: nivel físico.

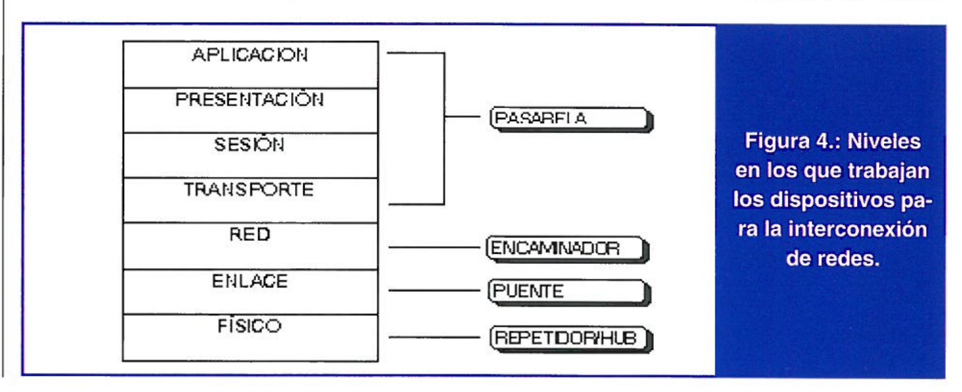


Figura 4.: Niveles en los que trabajan los dispositivos para la interconexión de redes.

DISPOSITIVO	NIVEL AL QUE TRABAJA
Repetidor	físico
Concentrador	físico
Puente	enlace
Encaminador	red
Pasarela	aplicación

Tabla 3.: Niveles de operación de los diferentes dispositivos interconectantes.

MEDIO FÍSICO	DISTANCIA
Cable telefónico	1.818 metros
Cable coaxial grueso	500 metros
Cable coaxial delgado	185 metros

Tabla 3.: Niveles de operación de los diferentes dispositivos interconectantes.

• Los paquetes recibidos en ellos pasan a la velocidad de la red, es decir no incorporan retardos en la retransmisión.

En cierta manera, se puede considerar a un repetidor como un dispositivo que puede segmentar una red; sin embargo esto no es del todo cierto: como ya se ha explicado, segmentar una red significa dividir de alguna forma el tráfico de ésta, de tal manera que las estaciones de un segmento sólo "vean" la información que llega del otro segmento que tiene como destino alguna estación de este segmento. Un repetidor no es capaz de realizar dicho trabajo, lo único que puede hacer es dejar pasar la señal siempre, independientemente y sin tener en cuenta (porque no sabe hacerlo) el segmento de red al que vaya dirigida la comunicación. Globalmente se debe entender que un repetidor permite incrementar la distancia entre estaciones de una red, e incluso unir dos tramos de una red que se encuentra distantes.

La tabla 4 refleja varios ejemplos de distancias en diferentes medios de transmisión a partir de las cuales y si se sobrepasan es necesaria la instalación de un repetidor.

A modo de resumen, se puede decir que los repetidores se utilizan:

• Cuando se quiere sobrepasar la distancia impuesta por la topología entre dos estaciones.

• Cuando el medio que rodea la red ofrece un alto grado de interferencias.

HUB

Los hubs son dispositivos creados para esquematizar las conexiones de una red. Se utilizan fundamentalmente, para crear topologías físicas en estrella que implementan topologías lógicas en bus, para aprovechar las ventajas de una topología en estrella. Los hubs internamente conectan sus puertos de entrada formando un bus lineal, para que de esta forma pueda implementarse el bus logicamente.

Un hub consiste en una caja sobre la que se centralizan todas las conexiones de una red. El número de puertos de un hub, es decir, de posibles conexiones, es variable; oscila entre los 8 y 12, por eso, si la red tiene más estaciones es necesario instalar más de un hub e interconectarlos entre ellos, como muestra la figura 3. Además, los hubs suelen incorporar un puerto para conexiones con un soporte central con cable coaxial o fibra óptica.

Existen dos tipos de hubs: los activos y los pasivos. Los hubs pasivos unicamente funcionan como centro de conexiones; los hubs activos, además, incorporan la función de repetidor.

Las ventajas que la incorporación de un hub añade son muchas: además de las agregadas por una topología en estrella (fácil detección de un error en la red, autonomía de las estaciones, ...), las suyas propias: facilidad de conexión y desconexión de estaciones, centralización del cableado, etc.

En la actualidad la practica totalidad de las redes Ethernet que se instalan se hacen utilizando un hub.

En ocasiones, un hub puede quedar reducido a un placa con sus puertos para poder conectar las estaciones; esta placa puede introducirse en un servidor; dependiendo del sistema operativo de red que se tenga, existen utilidades de administración de las placas hubs.

También es posible albergar varias placas hubs, junto a otros elementos (puentes, bakcbones...) en una caja de conexiones (wiring centers); estas cajas son, además del soporte para la conexión de diferentes placas, un centro de control que monitoriza y gestiona el cableado de la red.

Además de la clasificación inicial de los hubs, se puede hacer una nueva clasificación lógica atendiendo básicamente a la configuración del cableado estructurado; esta clasificación divide a los hubs en tres grupos: hubs para un grupo de trabajo, hub intermedio y hub corporativo. Esta división permite estructurar las conexiones de una red distribuida por un edificio. El primero de ellos es un hub al que se conectan las estaciones próximás (por ejemplo reune todas las conexiones de una habitación); el segundo es un hub que centraliza las conexiones de los hubs para un grupo de trabajo (por ejemplo reune todas las conexiones de una planta), el tercero hace referencia a hubs que centralizan todas las conexiones de la red; suele tratarse de una caja de conexiones que permite la conexión de placas diferentes, no sólo placas de hubs, (alberga placas de diferentes tipos de redes, token-ring y ethernet, puentes y encaminadores que las unen, etc) y que suele tener una conexión con una línea de alta velocidad, etc. La figura 5 muestra una interconexión de hubs de los tres tipos.

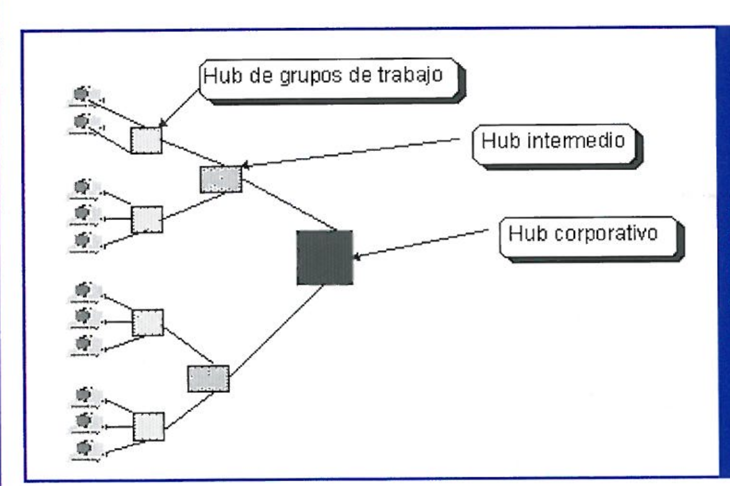


Figura 5.: Método de conexión de hubs según clasificación.

PRÓXIMO NÚMERO

En el siguiente número se terminará de describir los dispositivos que permiten la interconexión y expansión de una red: MAU, bridge, router, backbone y gateways.

LAS BARRAS DE COPPER

Pedro Antón

Una vez más, la paleta es lo más importante de este efecto. A lo largo de este curso se ha visto, que la paleta es algo esencial a la hora de crear un efecto gráfico, la modificación de ésta dentro de un mismo código, puede crear resultados visuales, totalmente distintos. En este artículo, se explicará uno de los efectos más sorprendentes del mundo de las demos, las barras de copper. El código para crear este efecto es realmente simple,pero el resultado obtenido es algo que no pasará inadvertido fácilmente.

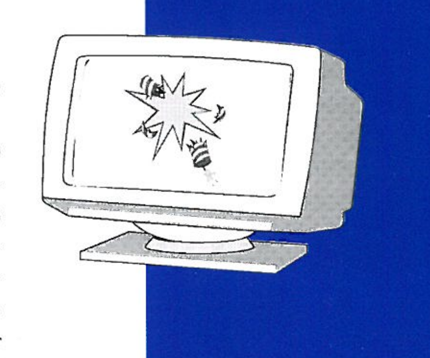
Aún recuerdo, una discusión que mantuve hace unos años, con un gran amigo mío, sobre la posibilidad de generar una gran cantidad de colores en modo texto. El origen de la discusión fue un programa que llegó a nuestras manos, el cual, parecía tener un fantástico degradado como fondo, y aquello parecía ser modo texto. Pero aquello no era todo, además realizaba un suave scroll al pulsar los cursores, en fin, algo realmente impresionante. La solución, es muy sencilla. Efectivamente, aquello era modo texto, el degradado no era ni más ni menos, que las barras de copper, y el scroll, no era más que, la modificación de la dirección de comienzo de la VGA, algo que se verá próximamente en este curso.

LA VGA

La VGA es el principal problema de las aplicaciones gráficas.

La aparición de las tarjetas de vídeo MCGA y VGA aumentó considerablemente la capacidad gráfica de los PC´s, se consiguieron, mayores resoluciones y la señal producida por la tarjeta pasó a ser analógica, de esta forma, el rango de colores a representar es ilimitado, dependerá de la calidad del DAC (Digital Analog Converter) que use el sistema de vídeo.

Un DAC, o convertidor digital- analógico es un dispositivo electrónico, que se encarga de convertir una señal de entrada digital en analógica. Es fácil de intuir, que la calidad de la salida dependerá en gran parte del número de bits usados, para esta conversión. El DAC que usa una tarjeta VGA standard permite visualizar 256 colores de una paleta de 262.144. Aunque la mayoría de las tarjetas actuales, permiten visualizar 16 millones de colores simultáneamente, esto se consigue usando un DAC de 24 bits, es decir, un byte, por cada una de las compo-



Seguimos avanzando y estudiando nuevos efectos para añadir a nuestras demos. Las famosas barras de colores, moviéndose en el fondo de una imagen, no son más, que rápidas variaciones del color de fondo. Este mes en Cómo programar una Demo las barras de Cooper

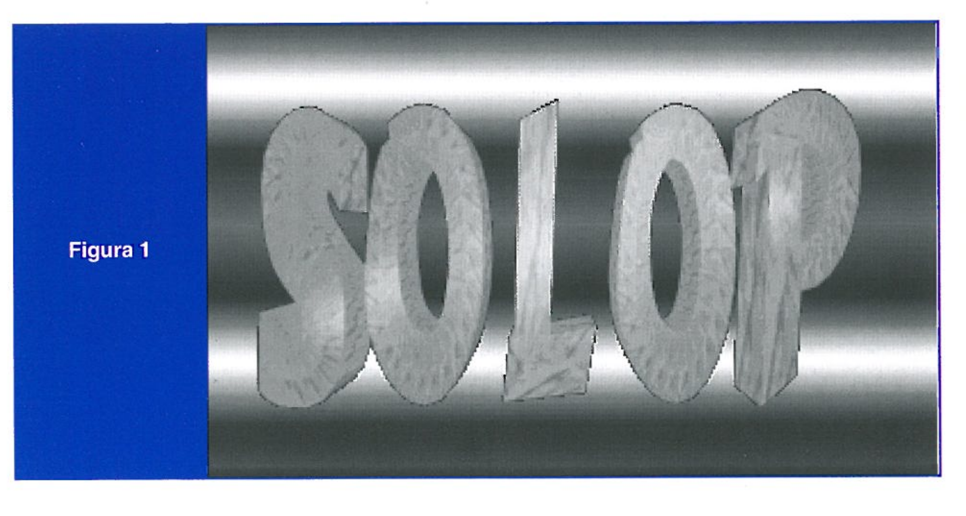


Figura 1

nentes de color Red, Green, Blue (rojo, verde, azul).

Pero antes de entrar en detalle, sobre cómo funciona el DAC de la VGA, veamos el esquema básico de un sistema de vídeo (Figura 2), para a continuación comentar algunos de sus componentes.

Buffer de video
Decodificador de attr
Generador de señal de video
Generador de caract.
PANTALLA
CRTC
Control de modo

Figura 2

EL BUFFER DE VÍDEO

La memoria más lenta del ordenador es la del sistema vídeo.

Sobre esta parte de la VGA se ha hablado ya en anteriores artículos, por lo tanto, no se entrará en muchos detalles. Como ya se ha visto a lo largo del curso, en el modo de vídeo 13h, esta zona de memoria de encuentra ubicada en la dirección de memoria A000h, el lector conoce sobradamente, el principal problema de buffer de vídeo: los accesos a la memoria de vídeo son mucho mas lentos que los accesos a memoria convencional.

No obstante se debe destacar en esta apartado, que el sistema de vídeo, es el encargado de leer continuamente lo datos de este buffer, refrescando la pantalla. Dependiendo del modo de trabajo la pantalla se refresca, entre 50 y 70 veces, esta dependencia nos permite intuir, la posibilidad de modificar dicho valor por software.

CONTROLADOR DEL TUBO DE RAYOS CATÓDICOS

El CRTC se encarga de sincronizar los datos almacenados en el buffer con la pantalla.

El controlador del tubo de rayos catódicos, se denomina de forma abreviada CRTC (Catode Ray Tube Controler) y es el encargado de generar las señales de sincronismo horizontal y vertical, así como de sincronizar la lectura de los datos del buffer con la señal de barrido. Además es el encargado de seleccionar la parte del buffer de vídeo que se ha de presentar en pantalla, detectar la posición del lápiz óptico, y otra serie de funciones en las que no entraremos en detalle.

Como programar en CRTC, será visto en un artículo posterior a este.

EL MONITOR

Actualmente, la inmensa mayoría de los monitores que existen en el mercado son analógicos, por ello, únicamente se hablará de este tipo de monitores en este artículo.

La imagen de un monitor, está formada por un grupo de líneas horizontales denominadas traza (raster). A este tipo de monitores se les denomina dispositivos de barrido por traza (raster-scan). Básicamente un haz de electrones barre cada traza de izquierda a derecha, comenzando por la esquina superior izquierda de la pantalla. Este haz de electrones, consta en realidad de tres haces separados. Cada uno de ellos controla uno de los colores primarios de vídeo (rojo, verde y azul). Por lo tanto, cada pixel de la pantalla, consta en realidad de tres puntos.

EL COLOR

El número y variedad de colores que pueden ser visualizados en un modo de vídeo dependen del diseño de la decodificación de atributos del modo de trabajo y del generador de señales de vídeo (DAC). Estudiaremos el caso de un DAC estándard, las tarjetas de vídeo que incorporan DAC más avanzados, por regla general, mantienen compatibilidad con las VGA estándar.

La paleta es algo esencial a la hora de crear un efecto gráfico

En modo gráfico, el decodificador de atributos, no usará los registros de paleta, haciendo más cómodo el proceso de generación de color. Pero si se hará necesario, como bien sabemos, modificar los colores del DAC a nuestro antojo, asignando a cada uno de los posibles 256 valores de la paleta, un valor de RGB entre 1 y 64, esto

LISTADO 1

```
PROCEDURE PutColor (Color,R,G,B:Byte);
assembler;
Asm
  mov dx,03C8h
  mov al,Color
  cli
  out dx,al
  inc dx
  mov al,R
  out dx,al
  mov al,G
  out dx,al
  mov al,B
  out dx,al
  sti
End;
```

```
PROCEDURE GetColor (Color:Byte;Var
R,G,B:Byte);
assembler;
Asm
  mov dx,03C7h
  mov al,Color
  cli
  out dx,al
  mov dx,03C9h
  in al,dx
  mov BYTE PTR R,al
  in al,dx
  mov BYTE PTR G,al
  in al,dx
  sti
  mov BYTE PTR B,al
End;
```

da una posibilidad de 262.144 posibles colores. Entre la mayoría de los programadores noveles existe la creencia de que cada color está formado por 3 bytes completos, de esta forma se podría conseguir un degrado de cada color básico de 256 tonalidades, sin embargo eso es totalmente incierto, la cantidad máxima de tonalidades de un color básico (rojo, verde, azul) es de 64.

La operación de modificar un color, o la paleta completa, ya ha sido empleada, múltiples veces por nosotros, a lo largo de este curso, no obstante, recordaremos que los puertos de entrada salida 03C7h, 03C8h y 03C9h, son los encargados de realizar las operaciones de color, si bien, siempre existe la posibilidad de realizar esas operaciones por mediación de la BIOS del PC. En el Listado 1 se puede observar como modificar un color por mediación de sus componentes RGB, así como leer las componentes RGB de un determinado color de la paleta.

El efecto que se trata este mes depende principalmente de la modificación de un sólo color, el color del fondo de la pantalla. Pero, evidentemente se deberá tener en cuenta algún detalle más.

EL REGISTRO DE ESTADO DEL CRT

El puerto 3DAh nos informará del actual estado del tubo de rayos catódicos.

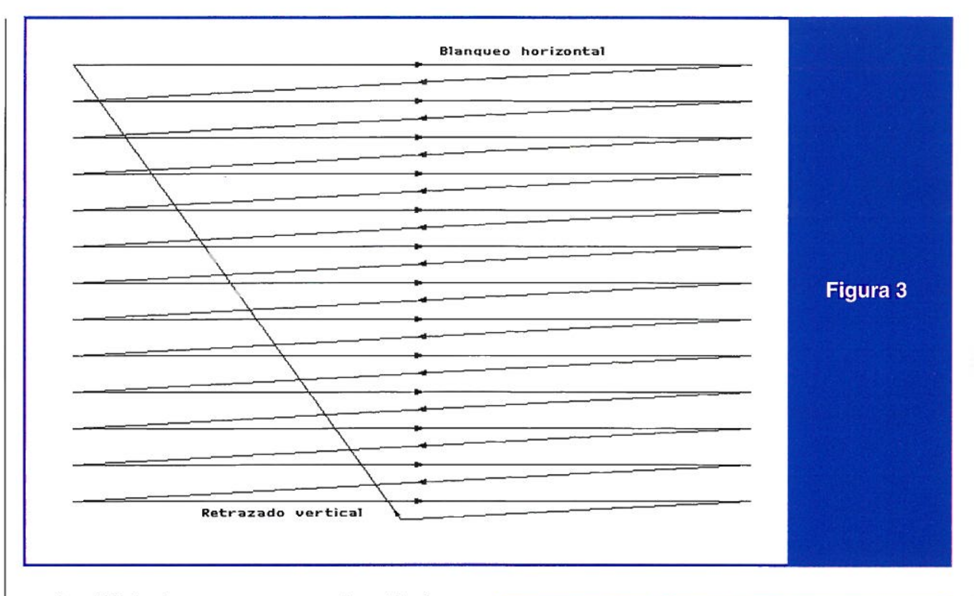


Figura 3

La VGA tiene un puerto de sólo lectura en la dirección 3DAh donde se puede ver reflejado el estado del de las señales de barrido. En nuestro caso, son de particular interés los bits 0 y 7, estos indican respectivamente los intervalos de blanqueo horizontal y el retrazo vertical, este último ha sido empleado en todos y cada uno de los ejemplos de este curso, aunque nunca había sido explicado en profundidad. Como se aprecia en el Listado 2, el bit 0 estará a 1 cuando se haya finalizado de escribir una línea horizontal en el monitor y el bit 7 estará a o cuando el controlador del tubo de rayos catódicos se encuentre realizando el retrazo vertical.

La aparición de las tarjetas de vídeo MCGA y VGA aumentó considerablemente la capacidad gráfica de los PC´s

LISTADO 2

```
PROCEDURE VerticalRetrace;
assembler;
Asm
  mov dx,3DAh
@@1:
  in al,dx
  test al,8
  jnz @@1
@@2:
  in al,dx
  test al,8
  jz @@2
End;
PROCEDURE HorizontalBlank;
assembler;
Asm
  mov dx,03DAh
@@1:
  in al,dx
  test al,01h
  jz @@1
End;
```

LISTADO 3

```
Procedure MakeCopper;
Const
Cnt       : Integer =0;
Deg       : Integer =0;
Angle     : Integer =0;
Pos       : Integer =0;
Var
VRetBool    : Byte;
Red       : Byte;
Green      : Byte;
Blue      : Byte;
Tecla      : Char;
SinusVal    : Integer;
Begin
Repeat
Cnt:=Pos;
Repeat
Red  := Colors[Cnt].R;
Green := Colors[Cnt].G;
Blue  := Colors[Cnt].B;
HorizontalRetrace;
PutColor (0,Red,Green,Blue);
Inc (Cnt);
If Cnt=Rasters Then Cnt:=0;
Asm
mov  VRetBool,0
mov  dx,03DAh
in   al,dx
test al,8
jz   @@NoVR
mov  VRetBool,1
@@NoVR :
end;
Until VRetBool=1;
Inc (Angle);
If Angle=1024 then Angle:=0;
SinusVal := FastSin(Angle);
Asm   mov   ax,SinusVal;sar   ax,6;mov
SinusVal,ax End;
   Pos     := Pos+SinusVal;
If Pos>=Rasters Then Pos:=Pos-Rasters;
If Pos<0       Then Pos:=Rasters+Pos;
Until Keypressed;
End;
```

LAS BARRAS DE COOPER

Cambiar el color del fondo en cada blanqueo horizontal generará las barras de copper.

Como ya se ha dicho, el CRTC se encarga de dibujar en el monitor el contenido del buffer de vídeo de forma correcta. Si se accede al buffer de vídeo a una velocidad mayor que el refresco de pantalla aparecerán efectos no deseados en esta. El motivo es evidente, adelantar al refresco, origina, la mezcla de dos o más imágenes.

En el listado 3 se puede observar el código necesario para generar las barras de copper. En él se observa cómo se produce el cambio del color 0 de la paleta en el momento justo de finalizar un blanqueo horizontal. También se puede ver cómo se realiza la operación de cambio de color en todo momento, excepto si el CRTC se encuentra realizando un retrazo vertical. El lector puede experimentar con el código y observar que eliminar esta espera producirá trastornos en el efecto, esto es debido a que el efecto básicamente consiste en cambiar el color del fondo en cada blanqueo horizontal, y el retrazado vertical, evidentemente, no tiene nada que ver con el blanqueo horizontal (ver figura 3).

Algo que llamará, sin duda la atención del lector es la inclusión de la unidad de 3D para este efecto. De esta unidad se emplea únicamente el seno, que será el encargado de producir una sensación de movimiento continuado de las barras de copper.

Pero además se observará como los valores que va a tomar el color 0, han sido previamente precalculados, esto es algo que se viene recalcando a lo largo del curso: "precalcula tantos valores como te sea posible". Bastará con echar un vistazo al Listado 4 para observar cómo se ha generado la tabla de color. Evidentemente la velocidad es algo realmente importante en este efecto.

El lector ya habrá intuido que el efecto también puede realizarse en modo gráfico.

Un DAC, o convertidor digital-analógico es un dispositivo electrónico, que se encarga de convertir una señal de entrada digital en analógica

LA UNIDAD DEMOVGA

Con esta explicación más profunda de la VGA, aunque no terminará aquí, la librería crece de nuevo añadiendo a esta las siguientes nuevas funciones:

HorizontalBlank
GetPalette
CopyColor
GetFadePalette
FadeUp
FadeDown

A continuación se explicará el funcionamiento y uso de cada una de ellas.

La función HorizontalBlank ya ha sido anteriormente explicada y se puede observar en el Listado 2.

BIBLIOGRAFÍA

Título: Sistemas de Vídeo.
Autor: Richard Wilton.
Editorial: Anaya.

Las funciones GetPalette, CopyColor y GetFadePalette son muy simples, de crear y de manejar. No obstante, es interesante destacar, que debido a que la rutina GetPalette está escrita en bajo nivel, requiere la dirección de memoria donde se va a almacenar dicha paleta. Así pues, para evitar posibles fallos, el lector debe asegurarse de haber reservado 768 bytes para almacenar los valores de RGB de cada uno de los 256 colores de la paleta. La unidad tiene una matriz de 768 bytes denominada AuxPalette donde se almacena la paleta cuando se llama al procedimiento GetFadePalette.

Los procedimientos FadeUp y FadeDown modifican los valores de RGB de la paleta actual hasta que estos se hacen cero, en el caso de la función FadeDown, o bien, son modificados, desde su estado actual, hasta el estado almacenado en el array AuxPalette. En el listado 5 se puede observar de forma más detallada como trabajan ambas funciones.

LISTADO 4

```
Procedure MakeColors;
Var
Cnt1 : Word;
Cnt2 : Byte;
Begin
Cnt1 := 0;
For Cnt2:= 0 to 63 do
Begin
Colors [Cnt1].R:=Cnt2;
Colors [Cnt1].G:=  0;
Colors [Cnt1].B:=  0;
Inc (Cnt1);
End;
For Cnt2:= 63 downto 0 do
Begin
Colors [Cnt1].R:=Cnt2;
Colors [Cnt1].G:=  0;
Colors [Cnt1].B:=  0;
Inc (Cnt1);
End;
For Cnt2:= 0 to 63 do
Begin
Colors [Cnt1].R:=Cnt2;
Colors [Cnt1].G:=Cnt2;
Colors [Cnt1].B:=  0;
Inc (Cnt1);
End;
For Cnt2:= 63 downto 0 do
Begin
Colors [Cnt1].R:=Cnt2;
Colors [Cnt1].G:=Cnt2;
Colors [Cnt1].B:=  0;
Inc (Cnt1);
End;
End;
```

TOWER ESTAMOS EN

SALON INTERNACIONAL DE LA INFORMATICA

Informat 96

BARCELONA, 30 SEPTIEMBRE - 6 OCTUBRE





Fira de Barcelona

Av. Reina Ma. Cristina 08004 Barcelona Tel. 233 20 00 Fax 233 23 19

SORTEOS DIARIOS EN NUESTRO STAND

INTRODUCCIÓN A LOS REGISTROS DE LA VGA

Santiago Romero, Miguel Cubas y Vicente Cubas

En el número anterior se estudio cómo inicializar y usar el modo X, pero sin adentrarnos en las posibilidades que nos ofrece la tarjeta VGA. En este número vamos a tomar el control nuestra tarjeta de una forma profesional gracias a los registros que la VGA usa para controlar sus parámetros internos

El PC es un ordenador extraño. En sistemas como el Amiga, por ejemplo, es indispensable el uso de los chips que la acompañan (tanto gráficos como sonoros) mediante ordenes hardware debidamente documentadas.

Los programadores del PC no disponen de estos chips ni de esta documentación y se tienen que arreglar con la utilización de los registros internos que la VGA usa para almacenar datos necesarios para su funcionamiento. Cosas como saber la localización del haz de electrones que redibuja la pantalla podría parecer algo meramente informativo pero puede ser usado para realizar vistosos efectos de paleta como las *Copper Bars* en modo texto (ver COPPER.EXE en los ejemplos del disco) y es indispensable para llevar una precisa sincronización y evitar parpadeos y "nieve" en las operaciones gráficas que requieran una gran velocidad.

Estos registros internos, que llevan parámetros importantes para la VGA, pueden ser accedidos mediante los puertos E/S a los que están conectados, que van desde el 3b0h al 3dfh y nos permiten su modificación y lectura, aunque aquí vamos a ver sólo los que nos interesan para la programación de los modos *unchained*. Hay que puntualizar que las tarjetas Super VGA disponen de nuevas funciones no estándar de control cuyo uso debe evitarse para asegurar la compatibilidad de nuestros programas en otras tarjetas, aunque aquí sólo se describirán aquellas funciones que puedan ser utilizadas en cualquier ordenador que disponga de una tarjeta VGA sin que haya ningún tipo de problema de compatibilidad.

LOS CONTROLADORES ASOCIADOS A LA VGA

Veamos en la tabla 1 los nombres de los componentes que son manejados mediante cada puerto, dándonos acceso a los registros de la VGA mediante los controladores de cada uno de ellos. En esta tabla todos los sufijos *_ADDR* son abreviatura de ADDRESS (dirección). Comencemos con la definición y explicación de cada uno de ellos:

Attribute Controller *(ATTR_ADDR, puerto 3c0h):* es el controlador de atributos, relacionado con todos los temas de colores, atributos, intensidad y parpadeo, paleta, etc... así como de control del color tanto del borde como de los planos y que está situado en el puerto 3c0h.

VGA Sequencer *(SEQU_ADDR, puerto 3c4h):* es el secuenciador de la VGA y maneja las funciones más críticas e importantes de la VGA tales como el reseteo de la tarjeta, el generador de carácteres, el *Memory Mode Register* (el cual contiene el bit CHAIN-4 que desencadenaba los 4 planos), etc... Su control se obtiene mediante el puerto 3c4h.

Graphics Controller *(GRAC_ADDR, puerto 3c3h):* es el controlador gráfico de la VGA, que estando situado en el puerto 3ceh da acceso a la lectura de planos (Ver GetPixelX( );), al *Miscellaneous Register*, al *Mode Control Register* y a funciones de operaciones lógicas y reseteo de registros.

Cathode Ray Tube Controller *(CRTC_ADDR, puerto 3d4h):* maneja todos los datos referentes al tubo de rayos catódicos que refresca o redibuja la pantalla, en un sistema asociado al monitor. Mediante este controlador, situado en el puerto 3d4h, tenemos acceso a información como el comien-

zo y fin del retrazo (tanto vertical como horizontal), el número de líneas a redibujar (*Total Vertical Register*), a la posición del cursor y a otras muchas funciones de pantalla. Este chip tiene una protección contra escritura que debe ser eliminada antes de modificar cualquier registro que sea accedido mediante el CRTC_ADDR o puerto 3d4h, aunque se pueden leer sus registros sin manipular la protección. La manera de eliminar la protección la veremos en la sección dedicada a la programación de los controladores.

VGA Enabler / Disabler *(VGAENABLE_ADDR, puerto 3c3h):* situado en el puerto 3c3h, el *least significant bit* (LSB = bit 0) de este puerto puede anular el acceso a los puertos VGA.

Miscellaneous Output *(MISC_ADDR, puerto 3c2h):* mediante este importante puerto, el 3c2h, pueden realizarse funciones de selección de polaridad, reloj, etc... Su uso lo encontramos, por ejemplo, en la rutina de inicialización del modo X, forzando al monitor a refrescar a 60 Hz mediante la utilización de este registro.

El ***SVGA Chip Controller*** *(CHIPSTECH_ADDR, puerto 3d6h)* no es estándar y por ello no va a ser usado para nuestros ejemplos, pero es interesante que los lectores conozcan su dirección E/S (3d6h) y que sepan que es el puerto mediante el que se accede a los registros específicos de cada tarjeta Super VGA.

PROGRAMACIÓN DE LOS REGISTROS DE LA VGA

La programación completa de la VGA requeriría una sección por sí misma y de hecho los lectores de Sólo Programadores ya disponen de la sección «Programación de la VGA» en algunos números de la revista, donde los registros son analizados con más complejidad y extensión, así que se van a introducir sólo sus fundamentos de una manera asequible y siempre pensando en los modos *Unchained.*

La programación de los registros es sencilla gracias a la conexión de estos a los puertos E/S, pero debido a la cantidad de registros y funciones de que disponemos hay que introducir un concepto denominado *índice de puerto* (*Port Index*).

ÍNDICES DE PUERTO

Para escribir o leer de un registro hay que saber el puerto en el que está y el Índice de puerto que tiene. El número de índice de puerto lo podríamos comparar al concepto de función de una interrupción. Como ejemplo, al imprimir textos mediante la interrupción 21h del MS-DOS escribimos:

```
MOV DX, offset Cadena
MOV AH, 9        ; número de función
INT 21h          ; llamada
```

Pero la misma interrupción la podemos usar para, por ejemplo, salir al MS-DOS:

```
MOV AH, 4Ch      ; número de
funciónINT 21h   ; llamada
```

El significado que tiene AH al llamar a la interrupción se puede comparar con el que tiene el índice de puerto, porque al igual que la interrupción 21h tiene diversas funciones, un puerto VGA puede controlar más de un registro, y necesitamos indicarle cuál de ellos queremos leer o modificar. Así pues, el índice de puerto le indica al puerto al que se escribe cuál es el registro que queremos modificar de todos los que ese puerto controla. Únicamente hemos de enviar los parámetros tal y como los requiere cada controlador, y ese es el principal problema de la programación a bajo nivel de la VGA.

En la VGA existen varias maneras distintas de leer o modificar los registros. Esto es debido al sistema que sigue cada periférico:

ATTRIBUTE CONTROLLER, ATTR_ADDR (3c0h)

- ESCRITURA:

Antes de poder acceder al Attribute Controller debemos limpiar un "*flip-flop*" interno de la VGA que controla la lectura y escritura. Esto se hace simplemente leyendo del puerto de *STATUS*, al que llamaremos *STATUS_ADDR*, situado en el puerto 3dah. Este registro se comentará al final de este artículo por su gran importancia. Después de leer del puerto de estado (STATUS_ADDR), se le envía al Attribute Controller el número de índice de puerto (el registro que se quiere leer o modificar) y más tarde el valor que se desea asignar al registro. De entre todos sus registros y funciones, el index port enviado nos asegura que el registro leído/modificado es el deseado.

```
inportb ( STATUS_ADDR );
outportb( ATTR_ADDR , index_port ); //
index port
outportb( ATTR_ADDR , valor );       //
valor
```

Después de cada acceso al Attribute Controller se desactiva el refresco de la pantalla y, como se puede ver en el **listado 2**, al finalizar las operaciones de escritura activamos de nuevo este refresco mediante:

```
outportb( ATTR_ADDR, 0x20 );
//reactivar refresco
```

- LECTURA:

Al igual que en la escritura, debe leerse primero del puerto de estado (3dah) y después enviar el índice de puerto al ATTR_ADDR. El valor del registro que queremos leer lo obtendremos del puerto ATTR_ADDR + 1:

```
inportb ( STATUS_ADDR );
outportb( ATTR_ADDR , index_port );
// enviar index-port
valor = inportb( ATTR_ADDR+1 );      //
leer valor
```

IDENTIFICADOR:	PUERTO:	NOMBRE:
ATTR_ADDR	3c0h	Attribute Controller.
SEQU_ADDR	3c4h	VGA Sequencer.
GRAC_ADDR	3ceh	Graphics Controller.
CRTC_ADDR	3d4h	Cathode Ray Tube Controller.
VGAENABLE_ADDR	3c3h	VGA Enabler/Disabler.
MISC_ADDR	3c2h	Miscellaneous Output.
CHIPSTECH_ADDR	3d6h	SVGA Specific Registers.

Tabla 1. Componentes controlados por la VGA.

Para probar el ATTR_ADDR disponemos de un ejemplo de cómo leer el color del borde de la pantalla VGA usando el Attribute Controller:

```
inportb ( STATUS_ADDR );          // limpiar flip-flop interno
outportb( ATTR_ADDR , 11h );      // función leer borde
color = inportb( ATTR_ADDR+1 );   // leer valor del puerto
```

Otra manera de no desactivar el refresco de pantalla cuando se accede al Attribute Controller es activar el bit 5 del índice, que si está a 0 hace que la VGA deje de redibujar la pantalla. Esto lo podemos hacer de muchas maneras, aunque la más sencilla es sumarle 32 al índice (El valor que indica el bit 5: 32 = 100000b). Si nos olvidamos de activar el bit 5 no ocurrirá nada gracias a la línea añadida al final de las funciónes *RegisterIn* y *RegisterOut* que asegura el correcto uso de este controlador, al igual que ocurre en el anterior apartado de escritura.

GRAPHICS CONTROLLER, GRAC ADDR (3ceh) CATHODE RAY TUBE CONTROLLER, CRTC_ADDR (3d4h) SVGA CHIPS CONTROLLER, CHIPSTECH_ADDR (3d6h)

- ESCRITURA:

En estos tres controladores la escritura se hace de similar manera. Se envía el Index Port (registro a modificar) al puerto correspondiente y se escribe el valor que se le desea asignar en el puerto asociado, PUERTO+1.

```
outportb( CRTC_ADDR, index );       // índice de puerto
outportb( CRTC_ADDR +1 , valor );   // valor a enviar
```

Estas dos llamadas a outportb también se pueden concretar en un solo outport gracias a que, como se comentó en el número anterior, outport(), outpw() y out dx, ax envían el byte bajo del valor de 16 bits que se le pasa al puerto especificado y después envían a Puerto+1 el byte bajo, de manera que pueden reducirse estas dos órdenes a:

```
outport( CRTC_ADDR, index | valor<<8 ); //enviar word
```

De esta manera, colocamos *index* en el byte bajo y *valor* en el byte alto (valor << 8) para que outport haga el trabajo en una sola orden. Mediante el OR (|) fundimos los dos bytes en un word.

Este sistema de programación de controladores debería sernos familiar ya que es el mismo que se usaba en el número anterior al seleccionar el plano de lectura de la VGA, controlado mediante la función 4 (Index port 4) del Graphics Controller:

```
outportb( GRAC_ADDR, 0x04 );          // índex 4 = selec. plano
outportb( GRAC_ADDR +1, ( x&3 ) );    // plano a leer
```

Este sistema es usado en muchas operaciones de inicialización, reseteo y control de la tarjeta, y nos permite controlar los 3 componentes de la misma manera, aunque hay que hacer notar que el CRTC Controller dispone de una protección contra escritura de registros que debe ser eliminada antes de intentar modificar cualquier registro que sea accedido mediante el puerto 3d4h. En operaciones de lectura no necesita ser eliminado. Su desprotección se realiza poniendo a 0 el bit 7 del índice o registro 11h, realizándose primero la lectura por el método ya descrito y después la desprotección. Este bit (bit 7) cuando está a uno impide la modificación de los registros y debe ser limpiado (bit 7 = 0) para realizar nuestras operaciones de escritura:

```
void UnProtectCRTC( void )
{
int v ;
outportb( CRTC_ADDR, 0x11 );              // índice = 11h
v = inportb( CRTC_ADDR + 1 );             // 3d5h
v = v & 0x7F;   // bit 7 = 0
outportb( CRTC_ADDR, 0x11 );              // índice = 11h
outport( CRTC_ADDR, 0x11 | ( v << 8 ) );  // desproteger
}
```

- LECTURA

La lectura es muy sencilla y consiste en envíar el índice de puerto al controlador correspondiente y leer del puerto contiguo, tal y como hace el siguiente código:

```
outportb( GRAC_ADDR, index );   // index port
v = inportb( GRAC_ADDR +1 );    // valor
```

VGA SEQUENCER, SEQU_ADDR (3c4h)

La programación del secuenciador de la VGA es igual que la de los tres anteriores salvo la excepción de escritura cuando el puerto de índice es 1. La función 1 significa la modificación del

ATTRIBUTE CONTROLLER	(ATTR_ADDR)	PUERTO: 3c0h
ÍNDICE:	**NOMBRE:**	**DESCRIPCIÓN DEL REGISTRO:**
10h	Mode Register. (Read/Write)	Bits: (los más importantes) -------- 0 Si es 1, modo gráfico. Si es 0, modo alfanumérico. 1 Modo monocromo si es 1. Modo de color si es 0. 2 Si 1, carácteres de 9 bits. 3 Si es uno, el bit 7 del byte de atributo significa parpadeo. Si es 0 = alta intensidad. 5 (VGA) Si 0, se ignora la comparación de línea. 6 (VGA) Si 1, anchura del pixel = 8. Para modos de 256 col.
11h	Border Color Register. (Read/Write)	Bits: -------- 0-5 Color del borde de la pantalla VGA.
12h	Color Plane Enable Register. (Read/Write)	Bits: -------- 0 Si vale 1, Plano 0 activado. 1 Si vale 1, Plano 1 activado. 2 Si vale 1, Plano 2 activado. 3 Si vale 1, Plano 3 activado.
13h	Horizontal Pixel Panning Register. (Read/Write)	Bits: -------- 0-3 Indican el nº de píxels a desplazar la pantalla hacia la izquierda. Ver tabla de HPPR.
14h	Color Select Register. (Read/Write)	Bits: -------- 0-7 Indican modo de acceso a la tabla DAC de la VGA.

VALOR	TEXT	256 color
0	1	0
1	2	n/a
2	3	1
3	4	n/a
4	5	2
5	6	n/a
6	7	3
7	8	n/a
8	0	n/a

Tabla HPPR.

Tabla 2. Registros del Attribute Controller.

GRAPHICS CONTROLLER	(GRAC_ADDR)	PUERTO: 3ceh
ÍNDICE:	**NOMBRE:**	**DESCRIPCIÓN DEL REGISTRO:**
05h	Mode Register. (Read/Write)	Para una completa descricpción del mode register, ver sección **Programación de la VGA** de Solo Programadores nº 20. Bits: (los más importantes) -------- 0-1 Modo de escritura. Controla cómo los datos de la CPU son transformados antes de ser escritos a la Video RAM: 0: El Modo 0 trabaja realizando escrituras en las que los datos pueden ser escritos de dos formas, comentadas en la Solo Prog. nº 20. 1: El Modo 1 es usado en transferencias de video a video. 2: El Modo 2 escribe un color a todos los pixels en el byte que escribamos en la Video RAM. El bit 0 es copiado en el plano 0, etc... 3: (VGA) El Modo 3 es usado para rellenar un area con un color o patrón. 3 Modo de lectura: 0: Los datos son leídos desde el plano seleccionado en el Read Map Select Register. (3CEh index 4). 1: Comparación entre los 8 pixels que ocupan el byte de lectura y el color del Color Compare Register (3CEh index 2). 6 Permite 256 colores si está a 1.
06h	Miscellaneous Mode Register. (Read/Write)	Bits: -------- 0 Indica modo gráfico si está a 1. Es un modo alfanumérico si está a 0. 2-3 Modo de memoria: 0: Usa A000h - BFFFh 1: Usa A000h - AFFFh Modos EGA/VGA. 2: Usa B000h - B7FFh Modos monocromos. 3: Usa B800h - BFFFh Modos CGA.

Tabla 3. Registros del Graphics Controller.

registro de polaridad del reloj de la VGA, y es un caso especial porque antes de modificar este registro se debe resetear el secuenciador. Esto se controlaría de la siguiente manera:

```
if ( index == 1 )
{
outport( SEQU_ADDR, 0x0100 );
outport( SEQU_ADDR, valor << 8 | 1 );
outport( SEQU_ADDR, 0x0300 );
}
else
outport( SEQU_ADDR, index | ( valor << 8 ) );
```

El sistema de lectura es igual que el de los tres anteriores sea cual sea el registro al que se desea acceder.

OTROS CONTROLADORES

El resto de los registros o controladores, tanto VGAENABLE_ADDR, STATUS_ADDR y MISC_ADDR como los que no han sido nombrados, se programan de la misma manera, la estándar del PC:

La modificación o lectura consiste en leer o enviar el registro directamente al puerto sin preocuparse del Index Port. Esto se hace cuando un puerto controla un solo registro, por lo que no puede existir ambigüedad sobre a cual nos referimos, que es como decir que ese puerto cumple una sola función. Son puertos de acceso directo porque el

Index Port es ignorado y nos ocupamos únicamente de escribir o leer en el puerto correspondiente. La mayoría de puertos del PC son usados de esta manera, más sencilla y simplificada:

LECTURA: valor = inportb(puerto);
ESCRITURA: outportb(puerto, valor);

En los *listados 1* y *2* podemos ver nuestras nuevas funciones de modificación de registros *RegisterIn* y *RegisterOut*, implementadas en la nueva librería *VGAREGS.H*, a modo de resumen sobre la programación de los diferentes componentes de la VGA. Además disponemos de funciones de lectura y escritura en puertos de manera que no haya que incluir la librería <dos.h> en nuestros programas.

LOS REGISTROS VGA

En las tablas 2, 3, 4 y 5 podemos ver los registros más interesantes en la programación del modo X, el significado de sus bits y el índice y puerto al que están asociados. En este número se incluyen los registros del Attribute Controller, Graphics Controller, VGA Sequencer y Miscellaneous Outport Register, además del STATUS_ADDR. Para el siguiente número dejamos el CRTC Controller, que nos permitirá hacer verdaderas "virguerías" con la VGA gracias a su control sobre todos los parámetros de retrazo, anchura y offset.

A continuación se describirá un registro con el que crearemos una *copper bar* como ejemplo de las posibilidades que nos brinda la programación de la VGA.

EL REGISTRO DE ESTADO STATUS_ADDR

Su descripción puede verse en la tabla 5. El significado de sus bits es sencillo de interpretar y proporciona acceso a los retrazos verticales y horizontales. Sobre el retrazo hay que saber que en el monitor hay un haz de electrones que bombardea el fósforo del monitor de manera que no se apague su tonalidad. Esto ocurre entre 60 y 70 veces por segundo, en las que el haz comienza a leer bytes de la VideoRam y a pintarlos empezando en la esquina (0,0), incluido borde, del monitor. Al final de cada línea horizontal el haz vuelve al principio de la línea siguiente. Este corto lapso de tiempo en que va de una línea a otra se llama *Retrazo Horizontal*.

SEQUENCER	(SEQU_ADDR)	PUERTO: 3c4h
ÍNDICE:	**NOMBRE:**	**DESCRIPCIÓN DEL REGISTRO:**
01h	Clocking Mode Register. (Read/Write)	Bits: (los más importantes) -------- 0 Si está a 1 los clocks de carácter serán de ancho = 8. 1 (EGA) Si está a 1 el CRTC usará sólo 2 / 5 de los ciclos de reloj. Si es 0, usará 4 / 5. 2 Si es 1, carga otro ciclo de reloj. 3 Si es 1, el Dot Clock es Master Clock / 2. Si 0 usa la misma frec. que Master Clock (3c2h bits 2-3). (Dobla pixels) 4 (VGA) Cada 4 ciclos de reloj carga los serializadores. Si es 0 lo carga cada ciclo. 5 Si es uno deja de refrescar la pantalla y le da a la CPU el control de todos los ciclos de reloj, pues ya no los usa.
02h	Map Mask Register. (Read/Write)	Controla los planos en los que se puede escribir. Podemos desactivar planos para impedir su modificación, etc... Bits: -------- 0 Permite escritura a plano 0 si está activado. 1 Permite escritura a plano 1 si está activado. 2 Permite escritura a plano 2 si está activado. 3 Permite escritura a plano 3 si está activado.
04h	Memory Mode Register. (Read/Write)	Bits: -------- 0 A 1 en modos alfanuméricos. 0 en modos gráficos. 1 Activado si hay más de 64k de memoria en el adaptador. 2 Si está a 1 activa modo de direcciones Odd/Even (impar-par). El modo Odd/Even coloca los bytes impares en los planos 1 y 3, y los pares en los planos 0 y 2. 3 (VGA) Si está activado los bytes 0 y 1 seleccionan planos de video memoria (modo de 256 colores).

Tabla 4. Registros del VGA Sequencer.

MISCELLANEOUS R.	(MISC_ADDR)	PUERTO: 3c2h
ÍNDICE:	**NOMBRE:**	**DESCRIPCIÓN DEL REGISTRO:**
01h	Miscellaneous Output Register. (Read: 3cch Write: 3c2h)	Bits: -------- 0 Si está a 1, emulación de color y Dirección Base = 3Dxh. Si está a 0, emulación Mono y Dirección Base = 3Bxh. 1 Si es 1, permite el acceso de la CPU a la VideoRam. 2-3 Modo de reloj de video: 0: 14 Mhz (EGA) 25 Mhz (VGA) 1: 16 Mhz (EGA) 28 Mhz (VGA) 2: Externo (EGA) Reservado (VGA) 4 (EGA sólo) Desactivar drivers internos si está a 1. 5 Seleccionar bancos de 64k altos si está a 1. 6 Horizontal Sync Polarity, negativa si está a 1. 7 Vertical Sync Polarity, negativa si está a 1. Estos 2 bits (6-7) indican el número de líneas de visualización. 0: 200 (EGA) Reservado (VGA) 1: 400 (VGA) 2: 350 (EGA) 350 (VGA) 3: 480 (VGA)

CRTC STATUS REGISTER	(STATUS_ADDR)	PUERTO: 3dah
ÍNDICE:	**NOMBRE:**	**DESCRIPCIÓN DEL REGISTRO:**
----	Input Status Register (Read only)	Bits (los más importantes): -------- 0 Si está a 1 el retrazo horizontal está en progreso. Si está a 0 el haz está redibujando la pantalla. 3 Si está a 1 está ocurriendo un retrazo vertical. Si está a 0 aún está refrescando la pantalla.

Tabla 5. Miscellaneous OutPut Register y Status Register.

El *Retrazo Vertical* es el período de tiempo en el que el haz vuelve a (0,0)



en diagonal desde la esquina inferior derecha tras haber terminado el refresco de todas las líneas de la pantalla y durante el cual los accesos a vídeo memoria no aparecen en pantalla hasta que los redibuje en el siguiente ciclo, cosa que debemos aprovechar para realizar las operaciones gráficas.

Como se puede ver en la tabla, el bit 3 cuando está a 0 nos indica que el haz de electrones está redibujando la pantalla y que no debemos realizar ninguna operación de escritura en VideoRam, por lo que debemos esperar a que esté a 1 para dibujar o volcar bitmaps y gráficos en pantalla.

Lo más práctico que se puede hacer es esperar a que esté a 0 (a que esté redibujando la pantalla), y luego esperar a que acabe de hacerlo (hasta que pase a estar a 1). Si sólo se comprobara esta última condición podríamos encontrarnos con el haz a punto de llegar a (0,0) con el consiguiente parpadeo en las operaciones gráficas. Veamos como sería el código de espera al retrazo vertical:

```
        mov dx, 3dah
vert1:
        in al,dx
        test al,8
        jnz vert1
vert2:
        in al,dx
        test al,8
        jz vert2
```

El retrazo horizontal lo controlamos de igual manera gracias al bit 0 de este registro. Mediante estos dos retrazos ya podemos evitar los parpadeos en nuestras operaciones gráficas y realizar efectos como el que vamos a ver a continuación.

UN EJEMPLO PRÁCTICO: LA COPPER BAR

Las *copper bars* se basan en los retrazos y en el cambio de paleta para cambiar el color del fondo y simular franjas horizontales. Para eso debemos saber cómo esperar un retrazo vertical y horizontal, cosas que ya podemos hacer gracias a nuestros nuevos conocimientos sobre los registros VGA y sobre el STATUS_ADDR.

LISTADO 1

```
unsigned char RegisterIn( int Controlador, char
index )
{
 unsigned char valor;

  switch ( Controlador )
  {
    case MISC_ADDR:
    valor = InPortb(0x3cc);
    break;
   // 0x3c2 es de escritura y 0x3cc de lectura

    case ATTR_ADDR:
InPortb(STATUS_ADDR); // reseteamos el flip-flop
OutPortb(ATTR_ADDR, index);
valor = InPortb(ATTR_ADDR+1);
break;

case SEQU_ADDR:
case GRAC_ADDR:
case CRTC_ADDR:
case CHIPSTECH_ADDR:
            OutPortb(Controlador, index);
            valor = InPortb(Controlador+1);
            break;

case VGAENABLE_ADDR:
default:    // resto de puertos
      valor = InPortb(Controlador);  break;
  }

 OutPortb( ATTR_ADDR, 0x20 ); // permitir refres-
co VGA
 return(valor);          // devolvemos el valor
}
```

Lectura de los registros VGA.

LISTADO 2

```
void RegisterOut( int Controlador, char index,
unsigned char valor )
{
  switch (Controlador)
    {
      case ATTR_ADDR:
    InPortb(STATUS_ADDR);   // resetear flip-flop
    OutPortb(ATTR_ADDR, index);
    OutPortb(ATTR_ADDR, valor);
    break;

      case SEQU_ADDR:
              if (index == 1)
        {
      OutPortw(SEQU_ADDR, 0x0100);   // resetear
el secuenc.
      OutPortw(SEQU_ADDR, valor<<8 | 1);
      OutPortw(SEQU_ADDR, 0x0300);
      break;
    };
case GRAC_ADDR:
case CRTC_ADDR:
case CHIPSTECH_ADDR:
   OutPortw( Controlador , index | valor<<8);
   break;

case MISC_ADDR:
case VGAENABLE_ADDR:
default:
   OutPortb(Controlador, valor);
   break;
}

OutPortb( ATTR_ADDR, 0x20 ); // permitir refres-
co VGA
}
```

Modificación de los registros VGA.

Para crear la Copper Bar se ha de seguir el siguiente algoritmo, que repitiéndose hasta que se pulse una tecla:

1. Esperar un retrazo vertical para cerciorarnos de que el haz de electrones está situado en la esquina superior izquierda de la pantalla.

2. Si la variable *Y* controla la altura de la barra en la pantalla de texto, esperarse *Y* retrazos horizontales significaría que el haz está situado ahora donde debe estar la primera línea de la barra. Lo hacemos mediante un bucle y pasamos al paso 3.

3. Para cada una de las líneas de la barra, cambiar la paleta del color de fondo con una tonalidad distinta de la anterior de manera que el haz dibuja la línea con el nuevo tono, pero como aún no ha llegado a redibujar la parte superior e inferior de la pantalla, estas líneas siguen con el color negro de fondo.

4. Incrementar o decrementar la variable *Y* según suba o baje la barra.

5. Cuando se hayan acabado todas las líneas de la barra, para que el rayo continue dibujando la pantalla con el color de fondo normal cambiamos la tonalidad para dejarla igual que el color de fondo, por ejemplo negro (0,0,0) y saltamos de nuevo al paso 1.

En los ejemplos del CD tenemos las nuevas funciones que nos serán útiles en cuestiones gráficas: *Ejemplo2.c* contiene la manera de cambiar la paleta utilizando el hardware de la VGA mientras que *Ejemplo3.c* y *Ejemplo4.c* contienen funciones de retrazos verticales y horizontales. El listado *Copperc.c* contiene una versión C de la Copper Bar, más asequible a los programadores que no dominan ensamblador.

EN LA PRÓXIMA ENTREGA

En la próxima entrega terminaremos la programación de los registros de la VGA mediante el CRTC Controller, y prepararemos nuestra librería modox.h para inicializar el modo X con distintas resoluciones de pantalla (1x4, 2x2, 4x1). Se incluirá además información extra sobre todos los demás registros de la VGA y una utilidad para examinarlos con exactitud.

GENERACIÓN DE CÓDIGO

Daniel Navarro

El momento en el que un compilador genera el código máquina parece ser el más fascinante de todos, hay muchas formas de realizarlo, y aquí se optará por una de las más sencillas de comprender

El compilador que se esta construyendo en esta serie está a punto de ser finalizado, pronto generará ejecutables, de hecho en este artículo se explica la parte más importante dentro de la generación de código, incluyendo la traducción de código EML a código máquina Intel 80x86.

En el último artículo se explicó todo lo relacionado con el análisis de expresiones, desde una discusión sobre el diseño de su sintaxis, hasta la generación de un código en lenguaje EML capaz de calcularlas (La sintaxis de letra y la descripción del lenguaje intermedio que se dieron en entregas anteriores de esta serie, se pueden encontrar en el disco de la revista como LETRA.BNF y LETRA.EML, en formato ASCII).

El compilador genera actualmente un listado denominado EML.LST con las instrucciones del lenguaje intermedio (LOC, STF, MUL, IGU, ...) correspondientes a las sentencias de asignación y las expresiones que aparezcan en el programa compilado. Durante el análisis del programa se utiliza la función generar() en puntos estratégicos (en los que hay generar las instrucciones de código máquina) para ir creando la traducción del programa. Esta función requiere el código de la instrucción a generar y el valor de su parámetro, en caso de que la instrucción lo requiera.

El listado generado no tiene utilidad ninguna dentro del proceso de compilación del programa, solo sirve para poder entender el proceso de generación de código y para comprobar que se esté realizando correctamente. No es de utilidad, porque no hay una máquina capaz de ejecutar "*LOC 2; LOC 3; SUM*"; estos mnemónicos han de ser traducidos a cadenas de código que sean ejecutables en un ordenador.

Cuando se introdujo el proceso de generación de código se explicó como se iba a hacer esto, por ejemplo, cuando sea invocada la función generar(*_loc,33*) esta emitirá, además del mnemónico de la instrucción en el listado EML, una cadena de bytes que cuando sea ejecutada en una máquina basada en un procesador Intel 80x86, hará lo que se entiende por un *LOC 33*, es decir, meterá la constante numérica 33 en la pila.

LA TRANSCRIPCIÓN A CÓDIGO MÁQUINA

Al comenzar la compilación se preparará, entre otras cosas, el vector mem[ ], éste no es mas que 64 kbytes que representarán la memoria de la máquina destino, es decir, un segmento con el programa compilado listo para ser ejecutado.

Tendremos una estructura de datos que defina cada una de las instrucciones del código intermedio, para cada instrucción; de este código se definen los siguientes campos:

• El mnemónico de la instrucción (una cadena de caracteres que contenga el nombre de la instrucción): este campo no sirve más que para generar el listado con las instrucciones EML.

• La cadena de código que implementa dicha instrucción sobre los procesadores 80x86: esta es una serie de bytes

formada por los códigos de operación de las instrucciones ensamblador necesarias para implementar la instrucción EML. Son necesarios unos conocimientos mínimos de ensamblador para poder construir esta cadena; aquí se dan todas las cadenas para EML, que son las que se necesitan para crear un compilador para Letra (ver cuadro 1).

• La longitud en bytes de de la cadena anterior.

• La posición del parámetro requerido dentro de la cadena de código, o bien un cero si la instrucción no requiere ningún parámetro.

Como un ejemplo podemos tomar la instrucción LOE que recibe como parámetro la dirección de una variable y apila su contenido. En ensamblador podemos hacer esto con la instrucción "*PUSH [dirección]*"" que tiene como código de operación "*FF 36 xx xx* " donde "xx xx" representan los dos bytes que ocupa el parámetro de la instrucción (la dirección de la variable). La longitud en bytes de la cadena LOE es 4, y la posición del parámetro 2.

Cuando durante la compilación del programa sea invocada la función *generar(_loe,direccion)* ésta escribirá la cadena de código anterior (FF 36 xx xx) en el vector mem[ ] a partir de la posición por la que se esté generando el código. El puntero que indique dicha posición será denominado "IP" (en referencia al registro "Index to Program"). Tras generarse una instrucción dicho puntero se avanzará el número de bytes que compongan la cadena de código emitida.

Las cadenas que se muestran en el cuadro 1 son las que utilizará el compilador de Letra para generar código. Estas son compatibles con todos los procesadores de la familia x86. Se pueden crear cadenas mas óptimas para los procesadores más modernos de dicha familia.

GENERACIÓN DE DATOS

Ir creando el código máquina es tan fácil como ir llamando a la función de generación de código indicándole que instrucciones de código intermedio tiene que generar.

CUADRO 1

Se describe tras el nombre de la instrucción y entre paréntesis la posición del parámetro dentro de la cadena de código, el parámetro se genera como un valor entero de 2 bytes sobre la cadena. Cuando el desplazamiento indicado para el parámetro es cero, es porque dicha instrucción no requiere ningún parámetro.

LOC (1):	b8 00 00 50
LOE (2):	ff 36 00 00
LOF (5):	5b d1 e3 ff b7 00 00
STE (2):	8f 06 00 00
STF (6):	58 5b d1 e3 89 87 00 00
MUL (0):	5b 58 f7 eb 50
DIV (0):	31 d2 5b 58 f7 fb 50
MOD (0):	31 d2 5b 58 f7 fb 52
SUM (0):	58 89 e3 01 07
RES (0):	58 89 e3 29 07
NEG (0):	89 e3 f7 1F
NOT (0):	89 e3 f7 17
AND (0):	58 89 e3 21 07
OR (0):	58 89 e3 09 07
DIS (0):	31 d2 5b 58 39 d8 74 01 42 52
IGU (0):	31 d2 5b 58 39 d8 75 01 42 52
MAY (0):	31 d2 5b 58 39 d8 7e 01 42 52
MAI (0):	31 d2 5b 58 39 d8 7c 01 42 52
MEN (0):	31 d2 5b 58 39 d8 7d 01 42 52
MEI (0):	31 d2 5b 58 39 d8 7f 01 42 52
IR (1):	bb 00 00 ff e3
IRF (6):	58 a8 01 75 05 bb 00 00 ff e3
LLA (1):	bb 00 00 ff d3
RET (0):	c3

Cadenas 80x86 para EML.

CUADRO 2

LOC: mov ax,0; push ax;
LOE: push [0];
LOF: pop bx; shl bx,1; push [bx+0];
STE: pop [0];
STF: pop ax; pop bx; shl bx,1; mov [bx+0],ax
MUL: pop bx; pop ax; imul bx; push ax;
DIV: xor dx,dx, pop bx; pop ax; idiv bx; push ax;
MOD: xor dx,dx; pop bx; pop ax; idiv bx; push dx;
SUM: pop ax; mov bx,sp; add [bx],ax;
RES: pop ax, mov bx,sp; sub [bx],ax;
NEG: mov bx,sp; neg word ptr [bx];
NOT: mov bx,sp; not word ptr [bx];
AND: pop ax; mov bx,sp; and [bx],ax;
OR: pop ax; mov bx,sp; or [bx],ax;
DIS: xor dx,dx; pop bx; pop ax; cmp ax,bx; jz eti; inc dx; eti: push dx;
IGU: xor dx,dx; pop bx; pop ax; cmp ax,bx; jnz eti; inc dx; eti: push dx;
MAY: xor dx,dx; pop bx; pop ax; cmp ax,bx; jle eti; inc dx; eti: push dx;
MAI: xor dx,dx; pop bx; pop ax; cmp ax,bx; jl eti; inc dx; eti: push dx;
MEN: xor dx,dx; pop bx; pop ax; cmp ax,bx; jge eti; inc dx; eti: push dx;
MEI: xor dx,dx; pop bx; pop ax; cmp ax,bx; jg eti; inc dx; eti: push dx;
IR: mov bx,0; jmp bx;
IRF: pop ax; test al,1; jnz eti; mov bx,0; jmp bx; eti:
LLA: mov bx,0; call bx;
RET: ret;

Instrucciones ensamblador correspondientes a las cadenas.

Cuando en el artículo del mes pasado se generaban las instrucciones de EML correspondientes a las sentencias de asignación, aún faltaba una información vital como son las direcciones donde serían ubicados (dentro del vector mem[ ]) los datos y las tablas. Evidentemente, no se puede generar código para "A=B+1" si no se sabe donde están las variables "A" y "B".

Los listados tienen una importancia vital dentro del desarrollo de un compilador, sirven para encontrar los errores del mismo

La forma de asignar memoria a los diferentes objetos varía mucho de unos lenguajes de programación a otros y de unos compiladores a otros. Letra es un lenguaje muy sencillo, porque sólo tiene variables de tipo entero con signo (2 bytes), y además como el compilador que se está construyendo generará programas COM donde la ejecución es dentro de un segmento de 64 kbytes, las direcciones (de etiquetas, datos o tablas) también serán del mismo tipo (2 bytes).

Los datos usados en un programa en Letra están declarados en una (o varias) zona de datos como la siguiente:

datos
a=10
b=20
tab1[10]
tab2[ ]=0:2:4:8
fin

Estas declaraciones podían venir en cualquier punto del programa, y por lo tanto, cuando se encuentra una declaración de datos se puede estar a medio generar el código máquina del programa en mem[ ], por lo que a los datos declarados se les irán asignando las próximas direcciones disponibles den-

tro del vector de memoria, reservando 2 bytes para cada dato, y para cada tabla 2 bytes por cada posición.

Para evitar que se ejecute una zona de datos se generará un salto inmediatamente antes de los datos al final de los mismos (se generará una instrucción "IR" de EML), pero hay un problema, ¿cómo sabe el compilador cuando comienza una declaración de datos en que dirección acabará?, sin dicha información no podemos generar el salto.

La solución a este problema consiste en reservar justo antes del primer dato, el espacio requerido para generar un "IR" y cuando termine la declaración de datos se generará dicho salto. Es decir, que no se genera el código de forma secuencial. A veces no se puede generar parte del mismo hasta que no se disponga de cierta información, como en este caso.

A la vez que se les van asignando direcciones a los datos se van inicializando las posiciones correspondientes (en el ejemplo anterior la posición de mem[ ] correspondiente a la variable "a" será inicializada con un 10), y en el caso en que no se indique el valor al que debe ser inicializada una variable, será inicializada a cero (esta es una práctica común en muchos compiladores).

Se puede encontrar la implementación de esto en el disco, para ello ver en el fuente *LETRA.CPP* el procedimiento *an_DECLARACION_DATOS()* .

ACCESO A OBJETOS ANTES DE DECLARARLOS

Una vez completada la información de la tabla de objetos, indicando la dirección que le corresponde a cada objeto, se puede ya poner para cada *LOE, STE, LOF y STF* (que son las instrucciones de lectura/escritura de datos y tablas) la dirección que requieren como parámetro (la del objeto que van a leer o actualizar), pero, ¿qué pasa cuando se tiene que generar una instrucción de acceso a un objeto, y éste aún no se ha declarado?, el compilador de Letra lo resuelve con el siguiente mecanismo (figura 1):

• Supóngase que se está generando código y vamos por la dirección IP=8300, en ese momento se tiene que generar un *STE* (para guardar el resultado de una expresión en un dato) y no se conoce su dirección (el registro del dato en la tabla de objetos indica "declarado=false"), entonces se genera la instrucción como si no pasara nada, inidicándose como parámetro la dirección del objeto, esta dirección será cero porque aún no se ha declarado el dato, y entonces como dirección del dato en su registro de la tabla de objetos, se indica la dirección en la que se ha puesto el cero en lugar de la dirección real del objeto.

Es decir, como la instrucción *STE* tiene la cadena 8f 06 00 00 con el parámetro en la posición 2 (en los dos últimos ceros), se generará dicha cadena entre las posiciones 8300 y 8303 (incluidas) y en el registro del objeto se indicará que su dirección es ahora la 8302.

• Si en este punto llegaran más accesos al mismo dato antes de su declaración, como por ejemplo otro STE en la dirección IP=9000, se hará lo mismo; se generará el STE como si no hubiera pasado nada (ahora se generará como si el objeto estuviera en la dirección 8302) y después se indicará como dirección del objeto la del último acceso, la 9002. De esta forma se van encadenando todos los accesos, en el registro del objeto se indica siempre la dirección del último acceso, y en cada acceso se indica la dirección del acceso anterior, hasta que llega el primer

El lenguaje EML no es ningún estándar, se ha inventado para generar código en el compilador de Letra

CUADRO 3

Debug es un ensamblador / desensamblador que acompaña a MSDOS desde sus inicios, se ha quedado algo anticuado, debido a que Microsoft no lo ha querido actualizar (tán sólo soporta el juego de instrucciones del 8086, ni siquiera el del 286). Para acceder a él solo se tiene que escribir "DEBUG" en la línea de comandos, entonces mostrará un guión, para indicarnos que está esperando un comando. Todos los comandos de debug están formados por una letra seguida opcionalmente de parámetros, para ver una lista de los comandos disponibles sólo se tiene que introducir como comando un símbolo "?" (cierre de interrogación). Para salir de debug, se utiliza el comando "Q" (quit).

Su utilidad reside en que permite ensamblar y desensamblar fácilmente. Ensamblar consiste en introducir una instrucción en ensamblador y debug generará el código máquina equivalente.

Por ejemplo, si se introduce "A 100" se indica que se quiere ensamblar una instrucción a partir de la dirección 100 hex (256 decimal), entonces debug mostrará "XXXX:0100" (las "X" hacen referencia a un número de segmento, este no tiene una especial relevancia). Ahora es cuando podemos introducir la instrucción a ensamblar, por ejemplo "mov bx,1234", con lo que debug mostrará "XXXX:0103" para darnos a entender que la instrucción anterior ocupó 3 bytes (0100,0101 y 0102) y que podemos ensamblar otra instrucción a partir de la dirección 103.

Si ya no se quieren introducir más instrucciones, se tiene que pulsar Enter y debug volverá a su línea de comandos. Para ver el código que ha generado la instrucción anterior podemos hacer "U 100" y debug mostrará:

```
XXXX:0100 BB3412 MOV BX,1234
XXXX:0103 ...
```

Luego los tres bytes formados por la instrucción anterior son (en hexadecimal) BB, 34 y 12.

Para introducir una cadena de bytes y hacer que debug muestre la instrucción (o instrucciones) correspondiente, se debe utilizar el comando "E dirección", por ejemplo si se introduce "E 100" debug muestra el byte que hay en dicha dirección y un punto:

```
XXXX:0100 BB.
```

Se van introduciendo bytes en hexadecimal, pulsando un espacio entre byte y byte, y cuando se pulsa Enter debug vuelve a su línea de comandos. Para ver la instrucción se vuelve a utilizar el comando "U" ("U 100" en este caso).

Notas sobre DEBUG.



acceso que tiene como dirección un cero.

• Cuando finalmente llega la declaración del objeto, se recorre la cadena de accesos, empezando por el último y hasta llegar al primero, poniendo ahora en todos la dirección definitiva del objeto. En ese momento se actualiza el registro del objeto poniendo también aquí su dirección, y se indica que ya ha sido declarado.

Se debe recordar que al acabar la compilación del programa se recorre toda la tabla de objetos para comprobar que ningún objeto se haya quedado sin declarar (lo que produciría un error de compilación).

LAS SENTENCIAS DE CONTROL DE FLUJO

El compilador ya genera código máquina capaz de evaluar las expresiones y las sentencias de asignación. A continuación se verá la forma de generar código para las sentencias de control.

Primero unas palabras para las etiquetas, estos objetos se tratan de forma similar a los datos, pero con las siguientes excepciones:

• Cuando se declara una etiqueta no se declara en una zona de declaración de datos, sino en cualquier otro punto del programa.

• A la etiqueta se le asigna también como dirección el valor de IP (el punto por que que estamos generando código), pero no se le reserva ningún espacio dentro de mem[ ] para la etiqueta, pues esta sirve para referirnos al código que viene inmediatamente a continuación, por lo que si se definen dos etiquetas seguidas ambas tendrán la misma dirección.

• El acceso a una etiqueta no se realiza por las instrucciones de acceso a datos (*STE, LOF, ...*) sino por las instrucciones: *IR* (salto incondicional), *IRF* (salto si es falsa la última expresión) y *LLA* (llamada a una función, que retornará con un *RET*).

• El acceso a etiquetas antes de que sean declaradas se resuelve exactamente igual que en el caso de los datos.

SENTENCIAS IR, HACER Y VOLVER

Estas sentencias de Letra son realmente fáciles de generar, todas tienen una instrucción de EML equivalente, por lo que cuando se analice en un programa una de estas sentencias, sólo se tiene que llamar a la función de generación de código e indicar qué instrucción de este lenguaje intermedio se tiene que emitir.

-IR se resuelve con un generar (_ir, dirección_etiqueta).
-HACER con generar (_lla, dirección_etiqueta)
-VOLVER con generar (_ret)

SENTENCIA REPETIR - HASTA

De todas las sentencias de control de flujo que implementan un bucle, esta es la más sencilla de implementar, se recuerda su sintaxis:

REPETIR
sentencias ...
HASTA (expresión)

Al comienzo de la sentencia, el procedimiento de análisis se queda con la dirección de memoria en la que comienza el bucle (*ip_inicio*). Tras terminar la sentencia se ha generado automáticamente el código para evaluar la expresión de *HASTA* (...); mientras dicha expresión sea falsa se debe repetir el bucle. Luego con generar (*_irf, ip_inicio*), se emitirá código para que se salte otra vez al principio de la sentencia si la expresión que se evaluó era falsa.

FIGURA 2

SENTENCIA	ETIQUETA	CÓDIGO QUE SE GENERA
repetir	IP_INICIO	
sentencias ...		CÓDIGO PARA SENTENCIAS ...
hasta (expresión)		CÓDIGO PARA EVALUAR EXPRESIÓN IRF IP_INICIO

Generación de código para MIENTRAS.

Para ampliar el lenguaje puede ser necesario ampliar el código intermedio, añadiendo nuevas instrucciones

SENTENCIA MIENTRAS - FIN

Esta sentencia es muy similar a la anterior, sin embargo al emitir código para la misma va a hacer falta generar dos saltos en lugar de uno. Se recuerda su sintaxis:

MIENTRAS (expresión)
sentencias ...
FIN

El procedimiento de análisis nuevamente se queda con la dirección de mem[ ] en la que comienza la sentencia en ip_inicio. Lo primero que se genera a partir de ip_inicio es el código para evaluar la expresión inicial. Mientras esta expresión sea cierta se debe repetir el bucle; luego tras evaluarla, si es falsa se debe saltar al final.

Aquí se plantea un problema similar al de las declaraciones de datos, se tiene que generar un IRF, pero aún no se puede saber a que dirección; nuevamente se reservará espacio para generar posteriormente dicho salto.

Al finalizar la sentencia, se tiene siempre que volver al principio para evaluar nuevamente la condición inicial, por lo que se hará con generar (_ir, ip_inicio). Tras la generación del IR final, ya se tiene en IP la dirección en la que acaba la sentencia, por lo que se generará el IRF que tiene que abortar el bucle cuando la expresión sea falsa.

SENTENCIA SI - SINO - FIN

La sentencia condicional es la más complicada, en cuanto a generación de

código, de todas las que dispone Letra. Puede adoptar dos formas, la más sencilla es:

SI (expresión)
sentencias ...
FIN

En este caso la resolución es la siguiente: tras haberse generado código para evaluar la expresión, se reserva espacio para generar un IRF (salto si no se cumple la condición) al final de la sentencia. Este salto se generará (como viene siendo habitual) al finalizar el análisis de la sentencia.

La otra forma que puede adoptar esta sentencia es:

SI (expresión)
sentencias...
SINO
sentencias...
FIN

La primera parte se resuelve igual: se evalúa la expresión y se reserva espacio para generar el IRF, aunque esta vez cuando la expresión sea falsa no se tiene que saltar al final de la sentencia, sino al segundo bloque de sentencias.

Al llegar la pieza *p_sino* (la parte "else"), se tiene que generar un salto incondicional (IR) al final de la sentencia (esto es para que no se ejecute el segundo bloque de sentencias si se ha ejecutado el primero), y vuelve a pasar lo mismo: como aún no se conoce la dirección se reserva espacio para generar este IR posteriormente. Tras reservar dicho espacio ya se sabe donde comienza el segundo bloque de sentencias, por lo que se genera el IRF inicial. Y al finalizar la sentencia se generará el IR que faltaba.

GENERACIÓN DE LISTADOS

Ahora el compilador ya genera una traducción del programa a código máquina en el vector mem[ ], pero para que ésta sea ejecutable faltan por resolver las llamadas a las funciones internas del lenguaje (borrar_pantalla, escribir, ...), esto se realizará en el próximo artículo.

Durante la compilación se va creando un listado en el fichero EML.LST, en él se registran una por una todas las instrucciones del código intermedio que

FIGURA 3

SENTENCIA	ETIQUETA	CÓDIGO QUE SE GENERA
mientras (expresión)	IP_INICIO	CÓDIGO PARA EVALUAR EXPRESIÓN
		IRF IP_FINAL
sentencias ...		CÓDIGO PARA SENTENCIAS ...
fin	IP_FINAL	IR IP_INICIO

se van generando. Cada vez que se llama a la función generar, ésta emite una línea con la posición de memoria actual (el valor de IP), el nombre de la instrucción generada y el parámetro (en caso de que haya).

Dicho listado no está ordenado, sino que muestra las instrucciones según se han ido generando, por ello se puede observar como, por ejemplo en una sentencia *MIENTRAS ... FIN*, la instrucción IRF es la última que se genera, aunque vaya en una posición de memoria anterior al resto de las instrucciones del bucle.

acceso anterior (si antes se hicieron otros accesos al mismo objeto). Estos accesos se corrigen antes de terminar la compilación (en cuanto se defina el objeto), pero en el listado quedan reflejadas las instrucciones según se generaron en un principio.

Si se finaliza con éxito la compilación (si el programa no tenía errores), el compilador grabará el código máquina definitivo a un fichero denominado "*CODIGO.X86*". Este fichero será el ejecutable en cuanto se añadan las funciones internas, de hecho ya es ejecutable si se renombra con extensión

Para generar código ejecutable sobre otro procesador, sólo se tienen que cambiar las cadenas de código de EML

También se puede observar que cuando se accede a un dato o etiqueta que aún no está declarado, se emite como dirección del mismo un cero (si es el primer acceso) o la dirección del

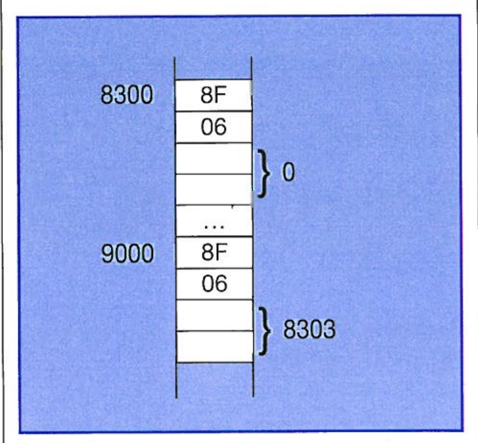


FIGURA 1: Acceso a objetos no declarados.

COM, siempre que en el programa compilado no se utilice ninguna función interna.

En el cuadro 2 se pueden observar las instrucciones del ensamblador 80x86 que corresponden a las cadenas de código de todas las instrucciones EML; para ver cual es el código de una instrucción ensamblador concreta, ver el cuadro 3.

Estas secuencias de instrucciones son las que se han generado en el fichero de código máquina. El compilador tras haber grabado dicho fichero, utiliza una herramienta del sistema llamada "debug" para que desensamble el fichero (para que vuelva a obtener el ensamblador a partir del código máquina).

Para que debug desensamble el programa compilado en Letra, se le tiene




que dar como entrada información sobre que zonas hay que desensamblar: sólo las zonas de código. Como el programa se carga a partir de la dirección 100 (en hexadecimal, ya que debug trabaja en dicha base), vendrá código máquina desde la dirección 100 hasta que se encuentre la primera declaración de datos. Tras los datos volverá a venir código hasta otra zona de datos o bien el final del programa.

Si por ejemplo se tienen que desensamblar las zonas 100-13A y 190-5C3, el fichero de órdenes para debug tendrá el siguiente aspecto:

```
U 100 13A
D 13B 189
U 190 5C3
Q
```

FIGURA 4

SENTENCIA	ETIQUETA	CÓDIGO QUE SE GENERA
si (expresión)		CÓDIGO PARA EVALUAR EXPRESIÓN
		IRF IP_SINO
sentencias ...		CÓDIGO PARA SENTENCIAS...
		IR IP_FINAL
sino	IP_SINO	
sentencias ...		CÓDIGO PARA SENTENCIAS ...
fin	IP_FINAL	

Generación de código para SI ... SINO.

Los conocimientos de ensamblador son necesarios si se quieren crear nuevas cadenas de código ejecutable

La orden "U" sirve para desensamblar, la orden "D" para hacer un volcado de memoria (esta se utiliza para ver el contenido de memoria en las zonas reservadas para datos), y por último la orden "Q" indica a debug cuando se ha finalizado ya el trabajo.

El compilador irá generando durante la compilación (según vaya sabiendo que zonas son de código y cuales de datos) estas órdenes en un fichero denominado "DEBUG.TMP", y al finalizar la compilación se invocará a debug con la siguiente línea de ordenes:

debug codigo.x86 <debug.tmp >codigo.lst

De este modo se generará en el fichero "CODIGO.LST" el desensamble del código máquina que ha generado el compilador.

En el listado "objetos.lst" (que mostraba el contenido de la tabla de objetos, con la información registrada sobre todos los datos, tablas y etiquetas que habían aparecido en el programa) se muestra ahora un campo nuevo: la dirección en la que ha sido ubicado cada objeto.

Se invita al lector a que pruebe a escribir algún pequeño programa y lo compile para estudiar después el resultado obtenido en los diferentes listados.

Se debe tener en cuenta un solo detalle: que el fichero con el código desensamblado ("CODIGO.LST") muestra todas las direcciones en hexadecimal, mientras que el resto de listados las muestran en decimal.

PRÓXIMO NÚMERO

El código que genera el compilador no es óptimo en absoluto, es más, es bastante malo. Esta forma de generar código es la más fácil de implementar (una cadena equivalente a cada instrucción de código intermedio), y en realidad tampoco es tan lento el código que se genera (por ejemplo el juego "columms.lt" va en un 8088 a 4´77Mhz prácticamente a la misma velocidad que en un Pentium). Pero un compilador comercial no puede permitirse generar el código de esta manera; se necesita una fase de optimización.

En el próximo número se resolverá lo que queda para finalizar el compilador, las funciones internas y el código "start up". Esta es una parte algo complicada, pero merecerá la pena el esfuerzo invertido para ver funcionar los programas compilados. Este artículo será el último de la serie "Construcción de un compilador", el número diez.

Pero no se va a dejar abandonado a Letra, sino que se seguirá mejorando con otros artículos más avanzados que se están preparando sobre compiladores, donde se abordarán temas como, por ejemplo, la optimización de código.

LOS CONTROLES BÁSICOS

Juan Manuel y Luis Martín

Los controles son los elementos insertables en el formulario que permiten al usuario interactuar con la aplicación. En el presente artículo se analizan los controles básicos incluidos en Visual Basic, de forma que el lector podrá comenzar a realizar sus primeras aplicaciones

En líneas generales, la realización de aplicaciones en Visual Basic consiste en la adición de controles a un formulario, cada uno de ellos con una misión muy concreta. Estos controles constituyen el interfaz mediante el cual el usuario podrá comunicarse con la aplicación.

En la actualidad existen una gran cantidad de controles, cada uno de ellos con unas características propias, y capaces de realizar casi cualquier operación. Visual Basic incorpora una serie de controles que se corresponden con los controles estándar de las aplicaciones Windows. Estos controles son los botones de comando, los cuadros de texto, las etiquetas, etc.

En este artículo se describirán algunos de estos controles, haciendo especial incapie en sus propiedades específicas, así como las posibles misiones que puedan desempeñar dentro de una aplicación.

LOS BOTONES DE COMANDO

Posiblemente, uno de los controles más utilizados en las aplicaciones Windows sean los botones de comando. Este tipo de controles tiene el aspecto de botón o pulsador, y permite llevar a cabo una determinada acción cuando el usuario realiza un clic sobre ellos. Al realizarse el clic, el botón cambia de aspecto, simulando la pulsación del mismo.

El nombre de la clase a la que pertenecen es *CommandButton*. Cuando se van insertando botones de comando en un formulario, Visual Basic les asigna los nombres por defecto *Command1*, *Command2*, etc. La tabla 1 muestra las propiedades, métodos y eventos de estos controles.

Aunque este tipo de controles spuede utilizarse de diversas formas, su uso mas frecuente consiste en desencadenar una determinada acción cuando el usuario los pulsa. Para ello, se utiliza su procedimiento de evento *Click*.

Dentro de un formulario, es posible incluir varios botones de comando. Además, es posible indicar cual de ellos será el botón por defecto. De esta forma, cuando el usuario pulse la tecla [Intro] dentro de dicho formulario, se asumirá que el usuario ha realizado un clic sobre dicho botón. Para ello, bastará con poner su propiedad *Default* a True. Dentro de un formulario, sólo puede haber un botón por defecto.

Igualmente, es posible definir un botón de cancelación. De esta forma, cuando el usuario pulse la tecla [Escape], se asumirá que se ha realizado un clic sobre dicho botón. Para ello, bastará con poner su propiedad *Cancel* a True. Dentro de un formulario, sólo puede existir un botón de cancelación.

En el cuadro 1 se indican los pasos a seguir para realizar un ejemplo del funcionamiento de los botones de comando.

LAS ETIQUETAS

Cuando dentro de una aplicación Windows realizada con Visual Basic se desea mostrar texto que no pueda ser alterado por el usuario, se utilizan unos controles denominados Etiquetas. Se trata de simplemente de recuadros dentro de los cuales se visualiza un texto. Las etiquetas disponen de un gran número de formatos para la presentación del texto, incluyendo diferentes tipos de alineamiento, de letra, tamaños, colores, etc.

El nombre de la clase a la que pertenecen las etiquetas es *Label*. Por defecto, Visual Basic les asigna los nombres *Label1*, *Label2*, etc. El texto que se visualiza por la etiqueta debe indicarse en su propiedad *Caption*. La cantidad de texto visualizado depende de varios factores como el tamaño del control, el tipo y tamaño de la fuente utilizada, etc. De cualquier forma, existe la posibilidad de ajustar las dimensiones del control de forma automática para que se pueda visualizar el texto. Para ello, basta con poner la propiedad *AutoSize* a True.

Otra propiedad que interviene en la cantidad de texto que se visualiza es *WordWrap*. Si el valor de esta propiedad es True, las palabras no se cortan al terminar el espacio disponible, sino que se pasan completamente a la siguiente línea.

Además, el textopuede visualizarse alineado de diferentes formas. Para seleccionar el tipo de alineamiento debe acudirse a la propiedad *Alignment*. Un valor 0 (*Left*) en esta propiedad indica que el texto se alineará a la izquierda. Un valor 1 (*Right*) hará que el texto se visualice alineado a la derecha. Por último, un valor 2 (*Center*) hará que el texto se visualice centrado.

También es posible indicar si se desea que las etiquetas presenten borde o no. Un valor 0 (*None*) en la propiedad *BorderStyle* indica que la etiqueta no tendrá borde, mientras que un valor 1 (*Fixed Single*) indica que la etiqueta presentará un borde en forma de línea.

Otro aspecto personalizable consiste en indicar si las etiquetas dejarán ver lo que hay debajo de ellas o no. Un valor 0 (*Transparent*) en la propiedad *BackStyle* indica que el fondo de la etiqueta será transparente, permitiendo así visualizar lo que haya debajo de ella. Por el contrario, un valor 1 (*Opaque*) no permitirá visualizar lo que haya debajo.

Además, las etiquetas permiten pasar el foco a otros controles. Ello se debe a que algunos controles no disponen de propiedades en las cuales indicar una tecla rápida, por lo que habrá que utilizar una etiqueta que realice dicha operación. Las etiquetas no pueden capturar el foco, ya que su texto no puede ser editado por el usuario. Sin embargo, es posible que las etiquetas muestren un carácter de acceso rápido. Para ello, basta con poner su propiedad *UseMnemonic* a True. De esta forma, utilizando el carácter ampersand (&) es posible crear una tecla de acceso rápido, tal y como se vio en artículos anteriores. Cuando el usuario pulse dicha tecla rápida, el foco será pasado automáticamente al objeto que sigue a la etiqueta según el orden de tabulación (propiedad *TabIndex*).

El cuadro 2 muestra un ejemplo de utilización de etiquetas en una aplicación.

TABLA 1

Grupo	**Repertorio**
Propiedades	Appearance, BackColor, Cancel, Caption, Container, Default, DragIcon, DragMode, Enabled, Font,Height, HelpContextID, hWnd, Index, Left, MouseIcon, MousePointer, Name, Parent, TabIndex, TabStop, Tag, Top, Value, Visible, WhatsThisHelpID, Width
Métodos	Drag, Move, Refresh, SetFocus, ShowWhatsThis, Zorder
Eventos	Click, DragDrop, DragOver, GotFocus, KeyDown, KeyPress, KeyUp, LostFocus, MouseDown, MouseMove, MouseUp

Propiedades, Métodos y Eventos de los botones de comando.

LOS CUADROS DE TEXTO

Otra forma de visualizar texto en una aplicación consiste en permitir que el usuario pueda alterar dicho texto. Para ello, Visual Basic incorpora otro control, similar a las etiquetas, pero que a diferencia de estas, si permite que el usuario altere el contenido de las mismas. Se trata de los cuadros de texto. De esta forma, es posible permitir que el usuario introduzca texto en la aplicación.

Los controles de este tipo pertenecen a la clase *TextBox*, y los nombres que por defecto les es asigna Visual Basic son *Text1*, *Text2*, etc. La tabla 3 muestra las propiedades, métodos y eventos de este tipo de controles.

El texto que se visualiza se almacena en su propiedad *Text*. Cuando un control de este tipo no tiene el foco, se limita a visualizar el texto contenido en dicha propiedad dentro de un recuadro. Sin embargo, cuando estos controles tienen el foco, dentro de dicho recuadro aparece un cursor de texto y permite que el usuario teclee nuevo texto. Los cambios realizados por el usuario dentro del texto se verán automáticamente reflejados en la propiedad *Text* del control.

La mayoría de las propiedades de los cuadros de texto son similares a las de las etiquetas, y realizan funciones idénticas. Este es el caso de las propiedades *Font*, *BackColor*, *ForeColor*, *Alignment*, *BorderStyle*, etc. Sin embargo, existen algunas otras propiedades específicas de este tipo de controles.

La propiedad *MaxLenght* indica el

TABLA 2

Grupo	Repertorio
Propiedades	Alignment, Appearance, AutoSize, BackColor, BackStyle, BorderStyle, Caption, Container, DataChanged, DataSource, DataField, DragIcon, DragMode, Enabled, Font, ForeColor, Height, Index, Left, LinkItem, LinkMode, LinkTimeout, LinkTopic, MouseIcon, MousePointer, Name, Parent, TabIndex, Tag, Top, UseMnemonic, Visible, WhatsThisHelpID, Width, WordWrap
Métodos	Drag, LinkExecute, LinkPoke, LinkRequest, Move, Refresh, ShowWhatsThis, Zorder
Eventos	Change, Click, DblClick, DragDrop, DragOver, LinkClose, LinkError, LinkNotify, LinkOpen, MouseDown, MouseMove, MouseUp

Propiedades, métodos y eventos de las etiquetas.

número máximo de caracteres que puede contener el cuadro de texto. Los caracteres demás que se introduzcan serán truncados. Por defecto, esta propiedad contiene el valor cero, que indica que no hay límite de caracteres.

También es posible que el texto introducido exceda de los límites de visualización. En este caso, será necesario utilizar barras de desplazamiento que permitan al usuario moverse por todo el texto existente. Para incluir barras de desplazamiento en un cuadro de texto, basta con asignar el valor apropiado a la propiedad *ScrollBars*. Un valor 0 (*None*) indica que el cuadro no contendrá barras de desplazamiento. Un valor 1 (*Horizontal*) visualiza una barra de desplazamiento horizontal. Un valor 2 (*Vertical*) visualiza una barra de desplazamiento vertical. Por último, un valor 3 (*Both*) indica que se visualizarán ambos tipos de barras de desplazamiento.

Los cuadros de texto permiten visualizar el texto en una sola línea o en varias. En el primer caso, cuando se alcanza el final del cuadro de texto, el texto se irá desplazando hacia la izquierda. En el segundo, cuando se alcance dicho final del recuadro, se realizará un salto de línea. Para indicar el tipo de funcionamiento debe utilizarse la propiedad *MultiLine*. Un valor True indica que el texto se visualizará en varias líneas, mientras que un valor False indica que el texto se visualizará en una única línea.

Además, los cuadros de texto también pueden funcionar como etiquetas, es decir, visualizar texto sin permitir que el usuario altere el contenido del mismo. Para ello se utiliza la propiedad *Locked*. Un valor True en dicha propiedad no permite que el usuario altere el texto. A diferencia de la propiedad *Enabled*, con la utilización de esta propiedad, el usuario puede acceder al control (el control puede obtener el foco), de forma que pueda consultarse el texto que contiene.

Otra de las propiedades importantes de este tipo de controles es *PasswordChar*. Mediante la utilización de esta propiedad es posible crear cuadros de texto para la introducción de palabras de acceso o passwords. La forma de emplear esta propiedad es

TABLA 3

Grupo	Repertorio
Propiedades	Alignment, Appearance, BackColor, BorderStyle, Container, DataChanged, DataSource, DataField, DragIcon, DragMode, Enabled, Font, ForeColor, Height, HelpContextID, HideSelection, hWnd, Index, Left, LinkItem, LinkMode, LinkTimeout, LinkTopic, Locked, MaxLength, MouseIcon, MousePointer, MultiLine, Name, Parent, PasswordChar, ScrollBars, SelLength, SelStart, SelText, TabIndex, TabStop, Tag, Text, Top, Visible, WhatsThisHelpID, Width
Métodos	Drag, LinkExecute, LinkPoke, LinkRequest, Move, Refresh, SetFocus, ShowWhatsThis, Zorder
Eventos	Change, Click, DblClick, DragDrop, DragOver, GotFocus, KeyDown, KeyPressed, KeyUp, LinkClose, LinkError, LinkNotify, LinkOpen, LostFocus, MouseDown, MouseMove, MouseUp

Propiedades, Métodos y eventos de los cuadros de texto.

TABLA 4

Grupo	Repertorio
Propiedades	Alignment, Appearance, BackColor, Caption, Container, DataChanged, DataField, DataSource, DragIcon, DragMode, Enabled, Font, ForeColor, Height, HelpContextID, hWnd, Index, Left, MousePointer, Name, Parent, TabIndex, TabStop, Tag, Top, Value, Visible, WhatsThisHelpID, Width
Métodos	Drag, Move, Refresh, SetFocus, ShowWahtsThis, Zorder
Eventos	Click, DragDrop, DragOver, GotFocus, KeyDown, KeyPressed, KeyUp, LostFocus, MouseDown, MouseMove, MouseUp

Propiedades, métodos y eventos de las casillas de verificación.

CUADRO 1

1 Añadir al formulario dos botones de comando y dos imágenes, situando las imágenes superpuestas, de forma similar a como se muestra en la figura 1.
2 Especificar en tiempo de diseño las siguientes propiedades:

Form	Name	frmComando
Caption		Ejemplo de botones de comando
Command1	Name	cmdInterruptor
Caption		&Verde
Command2	Name	cmdSalir
Caption		&Salir
Image1	Name	imgVerde
Picture		ICONS\MISC\TRFFC10A.ICO
Visible		False
Image2	Name	imgRojo
Picture		ICONS\MISC\TRFFC10C.ICO

3 Escribir los siguientes procedimientos de evento:

```
Private Sub cmdInterruptor_Click()
'Cambiar el dibujo
 imgVerde.Visible = Not imgVerde.Visible
 imgRojo.Visible = Not imgRojo.Visible
 'Cambiar el texto del interruptor
 If imgVerde.Visible Then
   cmdInterruptor.Caption = "&Rojo"
 Else
   cmdInterruptor.Caption = "&Verde"
 End If
End Sub

Private Sub cmdSalir_Click()
 End
End Sub
```

4Salvar el formulario en el archivo *COMANDO.FRM* y el proyecto en el archivo *COMANDO.VBP*.

Los dibujos del semáforo rojo y verde se van visualizando y ocultando cuando el usuario hace *clic* en el botón del interruptor. Igualmente, el texto visualizado sobre dicho botón es cambiado en el procedimiento de evento para indicar la próxima acción que realizará el botón.

Ejemplo de utilización de botones de comando.

asignándole un carácter que será el que se visualice en lugar de los caracteres tecleados.

Dentro de los eventos disponibles en este tipo de controles cabe destacar uno específico de esta clase. Se trata del evento *Change*, que se produce cada vez que se altera el contenido de la propiedad *Text*, es decir, cada vez que cambia el texto.

SELECCIÓN DE TEXTO

Los cuadros de texto incorporan además una serie de propiedades que permiten realizar operaciones de selección de texto, así como las consiguientes operaciones de cortado, copiado y pegado del texto a través del portapapeles de Windows. Estas propiedades únicamente se encuentran disponibles en tiempo de ejecución.

La propiedad *SelStart* indica la posición de comienzo del texto seleccionado. Esta posición viene indicada como el número de caracteres que se encuentran a la izquierda del primer carácter seleccionado. Si no hay texto seleccionado, esta propiedad contiene un valor que indica la posición actual del cursor dentro del texto.

La propiedad *SelLenght* indica la longitud del texto seleccionado, es decir, el número de caracteres seleccionados dentro del texto. Finalmente, la propiedad *SelText* contiene una copia del texto seleccionado. Si no existe texto seleccionado, esta propiedad contiene una cadena vacía.

Por último, existe otra propiedad denominada *HideSelection*, que permite especificar si el texto seleccionado continuará visualizándose resaltado una vez que el control pierda el foco. Un valor True indica que el texto seleccionado dejará de visualizarse resaltado una vez que el control haya perdido el foco. Por el contrario, un valor False si permitirá que el texto seleccionado se visualice resaltado aunque el control haya perdido el foco.

CASILLAS DE VERIFICACIÓN

Cuando en una aplicación es necesario que el usuario introduzca un valor de tipo Si/No o Verdadero/Falso se utiliza un control denominado casilla de verificación. Este control está compuesto por un pequeño recuadro, acompañado de un texto informativo. Cuando el control se encuentra seleccionado, dentro del recuadro se visualiza el símbolo de marca.

El nombre de la clase de este tipo de controles es *CheckBox*, y Visual Basic les asigna por defecto los nombres *Check1*, *Check2*, etc. La tabla 4 muestra las propiedades, métodos y eventos de este tipo de controles.

Debido a la naturaleza de este tipo de controles, una de sus principales misiones es la de visualizar un determinado estado de activación o desactivación, permitiendo al usuario cambiarlo. El estado del control se indica en su propiedad *Value*. Un valor 0 (*Uncheked*) indica que el control no se encuentra seleccionado, un valor 1 (*Cheked*) que el control se encuentra activado. Por último, esta propiedad dispone de un valor especial, 2 (*Grayed*), que indica que ele control está deshabilitado, es decir, con su propiedad *Enabled* a False. El valor de esta propiedad cambia automáticamente cuando el usuario realiza un clic sobre la casilla o el texto que la acompaña.

Para indicar el texto que acompaña a la casilla, habrá que utilizar la propiedad *Caption* del control. Dicha propiedad funciona de forma idéntica a la de otros controles.

La propiedad *Alignment* indica el lugar en el que se visualizará la casilla con respecto al texto que la acompaña. Un valor 0 (*Left Justify*) indica que la casilla se visualizará a la izquierda

TABLA 5

Grupo	Repertorio
Propiedades	Appearance, BackColor, Caption, ClipControls, Container, DragIcon, DragMode, Enabled, Font, ForeColor, Height, HelpContextID, hWnd, Index, Left, MouseIcon, MousePointer, Name, Parent, TabIndex, Tag, Top, Visible, WhatsThisHelpID, Width
Métodos	Drag, Move, Refresh, ShowWahtsThis, Zorder
Eventos	Click, DblClick, DragDrop, MouseDown, MouseMove, MouseUp

Propiedades, métodos y eventos de los marcos.

CUADRO 2

1 Añadir al formulario una etiqueta y un botón de comando

2 Especificar en tiempo de diseño las siguientes propiedades:

Form	Name	frmEtiqueta
Caption		Ejemplo de etiquetas
Label1	Name	lblDemo
AutoSize		True
Caption		Sólo Programadores
BorderStyle		1 - Fixed Single
Font	Size	18
Bold		True
BackColor		&H00FF0000&
ForeColor		&H0000FFFF&
Command1	Name	cmdSalir
Caption		&Salir

3 Escribir los siguientes procedimientos de evento:

```
Private Sub lblDemo_Click()
  Dim Temporal
  Temporal = lblDemo.BackColor
  lblDemo.BackColor = lblDemo.ForeColor
  lblDemo.ForeColor = Temporal
End Sub

Private Sub cmdSalir_Click()
  End
End Sub
```

4 Salvar el formulario en el archivo *ETIQUETA.FRM* y el proyecto en el archivo *ETIQUETA.VBP*.

Ejemplo de utilización de las etiquetas.

del texto, mientras que un valor 1 (*Right Justify*) indica que la casilla se visualizará a la derecha del texto.

El cuadro 3 muestra un ejemplo de utilización de este tipo de controles en una aplicación.

LOS MARCOS

En muchas aplicaciones Windows, los distintos controles suelen agruparse en grupos funcionales. Para realizar esta agrupación debe utilizarse un control denominado *Marco*. Se trata de un control representando por un recuadro dentro del cual pueden englobarse otros controles. Su única misión es la de agrupar o contener otros controles. El nombre de la clase es *Frame*, y los nombres que asigna Visual Basic a estos controles por defecto son *Frame1*, *Frame2*, etc. La tabla 5 muestra las propiedades, métodos y eventos de los controles de este tipo.

TABLA 6

Grupo	Repertorio
Propiedades	Alignment, Appearance, BackColor, Caption, Container, DragIcon, DragMode, Enabled, Font, ForeColor, Height, HelpContextID, hWnd, Index, Left, MouseIcon, MousePointer, Name, Parent, TabIndex, TabStop, Tag, Top, Value, Visible, WhatsThisHelpID, Width
Métodos	Drag, Move, Refresh, SetFocus, ShowWahtsThis, Zorder
Eventos	Click, DblClick, DragDrop, DragOver, GotFocus, KeyDown, KeyPressed, KeyUp, LostFocus, MouseDown, MouseMove, MouseUp

Propiedades, métodos y eventos de los botones de opción.

La utilización más habitual de este tipo de controles es conjuntamente con los botones de opción, de forma que estos últimos resulten mutuamente excluyentes. De esta forma, cuando se selecciona uno de estos botones de opción, los demás quedarán deseleccionados de forma automática.

Para incluir controles dentro de un marco, será necesario seleccionar el control a insertar de la caja de herramientas para, posteriormente, dibujarlo dentro del marco. No es posible la inclusión de controles mediante la realización de un doble clic dentro de la caja de herramientas.

Si existe un control que no se encuentra incluido en un marco, y se arrastra dentro de él, el control seguirá sin pertenecer al marco, aunque se encuentre apilado sobre él.

Los marcos cuentan con la propiedad *Caption*, de forma que es posible indicar el título del marco. Además, la propiedad *Enabled* de los controles que se encuentren dentro de un marco dependerá del valor de la propiedad *Enabled* del propio marco. Si se deshabilita el marco, todos los controles que se encuentren incluidos en él también quedarán deshabilitados, no pudiendo capturar el foco.

CONCLUSIÓN

En artículo se ha comenzado a analizar el funcionamiento de los controles estándar disponibles en Visual Basic para realizar distintas operaciones. Se recomienda al lector que experimente con sus propios ejemplos, alterando los diferentes valores de las propiedades de los controles con el fin de experiementar y comprobar los diversos resultados. En el artículo del próximo mes se continuará analizando el repertorio de controles básicos proporcionados por Visual Basic.

CUADRO 3

Cuadro 3.- Ejemplo de utilización de cuadros de texto y casillas de verificación.

1 Añadir al formulario un cuadro de texto, cuatro casillas de verificación y un botón de comando,

2 Especificar en tiempo de diseño las siguientes propiedades:

Form	**Name**	**frmTexto**
Caption		Ejemplo de casillas de verificación
Text1	Name	txtDemo
Text		
MultiLine		True
Scrollbars	2 - Vertical	
Check1	Name	ckhNegr
Caption		&Negrita
Font	Bold	True
Check2	Name	chkCurs
Caption		&Cursiva
Font	Italic	True
Check3	Name	ckhSubr
Caption		&Subrayada
Font	Underline	True
Check4	Name	chkTach
Caption		&Tachada
Font	StrikeThrough	True
Command1	Name	cmdFinal
Caption		&Finalizar

3 Escribir los siguientes procedimientos de evento:

```
Private Sub chkNegr_Click()
  txtDemo.Font.Bold = Not txtDemo.Font.Bold
End Sub

Private Sub chkCurs_Click()
  txtDemo.Font.Italic = Not txtDemo.Font.Italic
End Sub

Private Sub chkSubr_Click()
  txtDemo.Font.Underline = Not txtDemo.Font.Underline
End Sub

Private Sub chkTach_Click()
  txtDemo.Font.Strikethrough = Not txtDemo.Font.Strikethrough
End Sub

Private Sub cmdFinal_Click()
  End
End Sub
```

4 Salvar el formulario en el archivo *TEXTO1.FRM* y el proyecto en el archivo *TEXTO1.VBP*.

Ejemplo de utilización de las etiquetas.

LA CLASE CWnd Y DERIVADAS

Jorge R. Regidor

La clase *CWnd* nos proporciona toda la funcionalidad necesaria para poder manejar las ventanas de Windows. Esta clase encapsula la mayoría de las funciones del API para manejo de ventanas, por lo que existen funciones miembro de esta clase con un nombre bastante familiar para los conocedores de la programación de Windows utilizando C con el API de Windows. Este paralelismo permite una mayor facilidad a la hora de intentar averiguar como se hace alguna operación, ya que existirá con bastante probabilidad de éxito una función con el mismo nombre, pero que por norma tiene menos parámetros debido a que estos datos forman parte de los datos privados de la clase *CWnd*. Un claro ejemplo de esto último es que al llamar a una función miembro de la clase *CWnd*, no es necesario incluir el típico parámetro *hWnd* que identifica la ventana, ya que el valor de este handle se encuentra almacenado en la variable miembro *m_hWnd* de cada uno de los objetos *CWnd*, lo que permite identificar cada un objeto de la clase *CWnd* con cada ventana Windows creada.

Hay que resaltar la diferencia existente entre la clase *CWnd* y las ventanas Windows. Los objetos de la clase *CWnd*, se crean y se destruyen con sus correspondientes constructores y destructores, mientras que la ventana se crea con la función miembro *Create* o *CreateEX* y se destruye con la función *DestroyWindow*, sin que por ello se destruya el objeto. El método lógico para crear una ventana es el siguiente, primero se crea el objeto *CWnd*. Posteriormente, se llama a la función *Create*, que permite crear la ventana "real". Para destruir la ventana se debe seguir el proceso inverso, llamar a la función *DestroyWindow*, y después destruir el objeto. Estas operaciones sólo serán necesarias realizarlas en el caso de que se quiera crear una ventana hija de la ventana principal.

La clase *CWnd* y el método de direccionamiento de mensajes proporcionado por las MFCs ocultan la función *WndProc*, conocida por los programadores del API y núcleo de todas las acciones a emprender en una ventana Windows. En lugar de esto, la clase *CWnd* nos proporciona las funciones miembro del tipo *OnMessage*. Estas funciones deben ser sobrecargadas para permitir introducir el código para atender cada mensaje recibido por la ventana. En este punto se recomienda utilizar la herramienta *WizardBar* o *ClassWizard*, que se encargan de incluir el código necesario para sobrecargar estas funciones y además nos indican el modo de llamar a la función miembro de la cual derivan.

CWND Y SUS CLASES DERIVADAS

La clase *CWnd* como tal muy raramente es utilizada en detrimento de sus clases derivadas, que proporcionan diferentes tipos de ventanas. Muchas de estas clases derivadas, a su vez se derivan en otras aún más especializadas. Además, y siempre que nosotros necesitemos añadir una nueva funcionalidad, debemos derivar hacia nuestra propia clase desde la clase madre más apropiada. El hecho de atender a un determinado mensaje de un modo particular, implica que la clase a utilizar debe sustituirse mediante una clase



Si decíamos en el anterior artículo que la clase *CWinApp* era importante, más aún lo es la clase *CWnd*, encargada de gestionar todas las posibles operaciones a realizar sobre todas y cada una de las ventanas de nuestra aplicación

derivada a esta última. Si por ejemplo, queremos crear un botón que al pulsarse de un pitido, la operación a tomar será la siguiente. Sabemos que un botón es una ventana especializada, en concreto un control estándar de Windows. Además sabemos que existe una clase llamada *CButton* derivada, por supuesto, de la clase *CWnd* y que encapsula los métodos de este control así como sus particularidades. Ahora bien, al tener que incluir una nueva función y por tanto especializar aún más el tipo de botón debemos hacer un subtipo, es decir derivar la clase hacia nuestro botón especial, llamado por ejemplo *CBeepButton*.

CREANDO UNA VENTANA HIJA

Para crear esta nueva clase las operaciones a realizar pasan por utilizar *ClassWizard*, que de nuevo una vez más nos va a hacer el trabajo sucio, permitiéndonos en unos segundos crear los cimientos de nuestra nueva clase. Para ello abrimos *ClassWizard* y nos situamos en el tab *ClassInfo*. Aquí pulsamos el botón *Add Class...*, y sobre el menú en *new...*, después de esto nos aparece el diálogo para pedirnos los datos de la nueva clase. En el campo *Name* se indica *CBeepButton*, en el campo Base class se selecciona CButton. Fruto de esto, Visual C++ nos sugiere el nombre *BeepButton.cpp*, aunque nosotros podemos seleccionar otro nombre. Además vamos a deshabilitar el campo para añadir al Component Gallery en la parte inferior. El resultado es que el fichero *BeepButton.cpp*, que contiene la arquitectura básica de nuestra clase, se añade al proyecto.

Para añadir el Beep del botón, abrimos el fichero *BeepButton.cpp* y mediante *WizardBar*, atendemos al mensaje *BN_CLICKED*.

```
void CBeepButton::OnClicked()
{
  MessageBeep(-1);
}
```

La clase CWnd nos proporciona toda la funcionalidad necesaria para poder manejar las ventanas de Windows

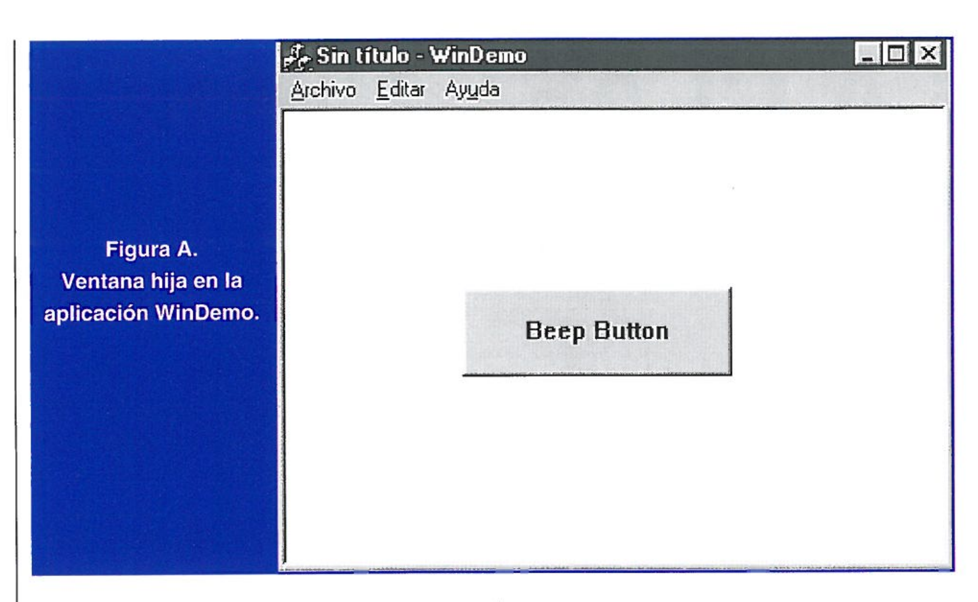


Figura A. Ventana hija en la aplicación WinDemo.

Con esto, ya hemos creado nuestra clase *CBeepButton* con la funcionalidad deseada, pero ahora debemos añadirla en una ventana.

Para ello se ha creado un proyecto con *AppWizard* llamado *WinDemo*, de tipo *SDI*, sin *ToolBar* ni *StatusBar* y con una vista basada en la clase CView. Nosotros vamos a incluir el botón dentro de la vista de nuestra ventana, por lo que abrimos el fichero *WinDemoView.cpp* y *WinDemoView.h*.

En la clase *CWinDemoView* vamos a añadir una variable miembro para que contenga la dirección al objeto *CBeepButton*, el cual será hijo de esta ventana.

```
class CWinDemoView : public CView
{
. . .

// Datos
public:
        CBeepButton
MyBeepButton;
};
```

Ahora debemos insertar el código necesario para crear el objeto *CBeepButton*, así como para crear la ventana, es decir, el botón. Para ello insertamos el siguiente código en el constructor de la clase *CWinDemoView* y la función miembro *Create*.

CWinDemoView::CWinDemoView()

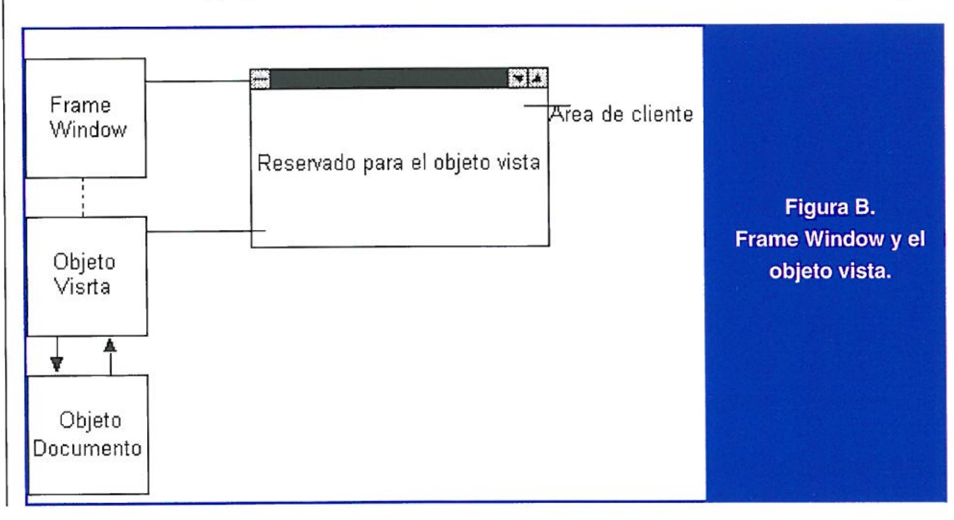


Figura B. Frame Window y el objeto vista.

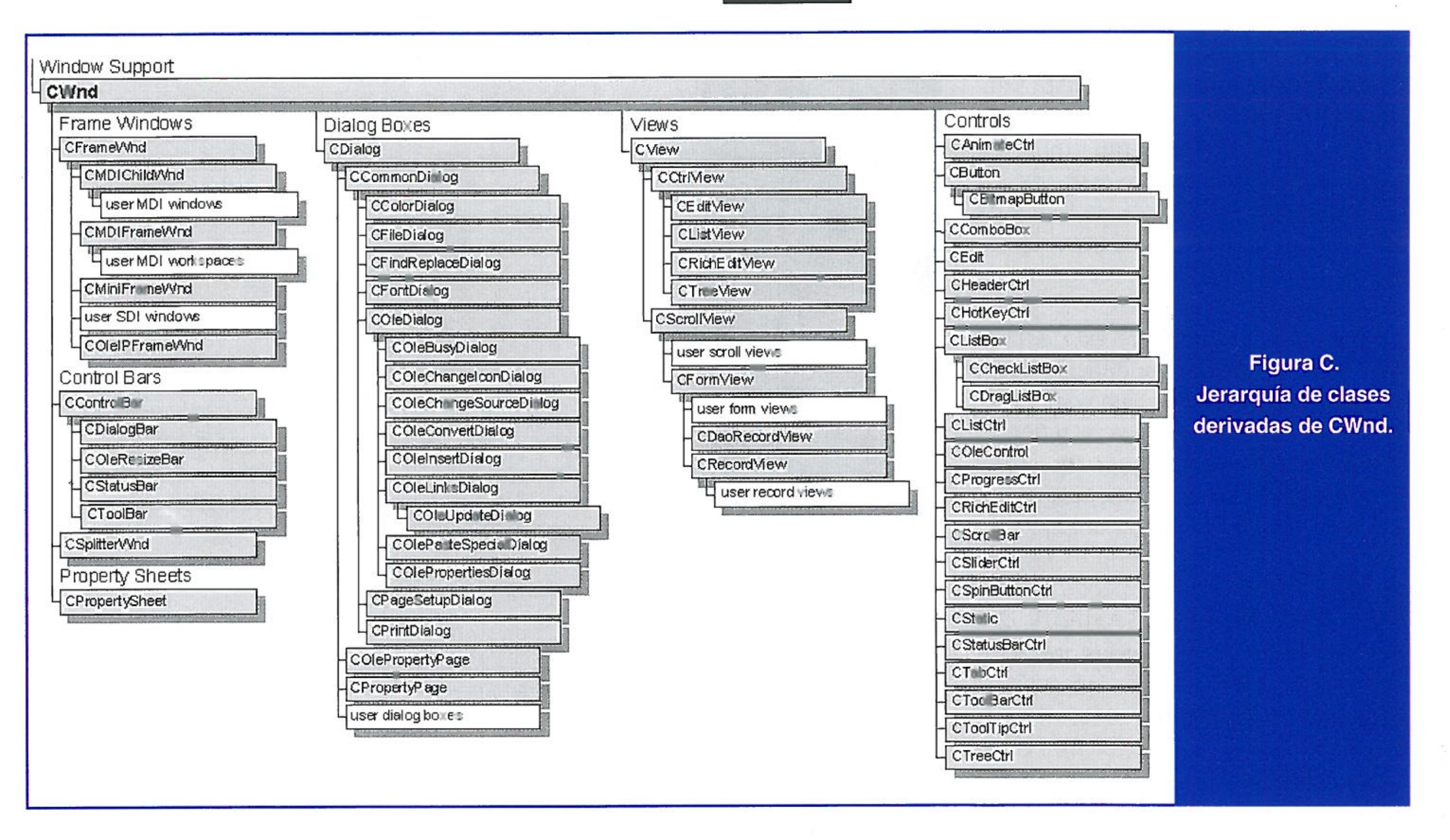


Figura C. Jerarquía de clases derivadas de CWnd.

```
 {
  MyBeepButton = new CBeepButton();
 }

BOOL CWinDemoView::Create
(LPCTSTR lpszClassName,
      LPCTSTR lpszWindowName,
DWORD dwStyle, const RECT& rect,
      CWnd* pParentWnd, UINT nID,
CCreateContext* pContext)
 {
  BOOLEAN bRes;
  RECT rcButton = {100,100, 250,150};

  bRes = CWnd::Create(lpszClassName,
lpszWindowName, dwStyle,
      rect, pParentWnd, nID, pContext);

      bRes &= MyBeepButton-
>Create("Beep Button",
      WS_CHILD | WS_VISIBLE ,
rcButton, this, 100);
  return bRes;
 }
```

En el constructor de la clase se crea el objeto mediante el operador new y se almacena el puntero en *MyBeepButton*. Posteriormente, y al crearse la vista (al llamarse a *Create* de la clase *CWinDemoView*), nosotros creamos el botón, y para ello llamamos a la función *Create* del objeto *MyBeepButton*, que tiene los parámetros de la función Create de la clase *CButton* de la cual deriva. Es importante insertar el estilo *WS_VISIBLE*, ya que si no, el botón será creado pero no se mostrará en pantalla.

Una vez concluido el programa, debemos destruir el botón y el objeto *CBeepButton*, y para ello debemos atender al mensaje *DestroyWindow* de la clase *CWinDemoView* y a su destructor, donde se debe eliminar el botón y el objeto asociado respectivamente.

```
      BOOL CWinDemoView::
DestroyWindow()
 {
  MyBeepButton->DestroyWindow();
  return CView::DestroyWindow();
 }

 CWinDemoView::~CWinDemoView()
 {
  delete MyBeepButton;
 }
```

La clase CWnd como tal muy raramente es utilizada en detrimento de sus clases derivadas

GRUPOS O TIPOS DE VENTANAS

Hay gran cantidad de clases derivadas directa o indirectamente de *CWnd*, divididas en varios grupos según su funcionalidad, por lo que vamos a dar un repaso a las clases más importantes con el objetivo de tener una visión global de ellas y así saber cual utilizarse en cada momento.

Existen 6 grupos de ventanas derivadas directamente de *CWnd*. Cada grupo tiene una misión específica y que se vuelve a especializar más aún en sus propias clases derivadas. Los grupos se describen en la Tabla 1.

FRAME WINDOWS

La ventana Frame es la ventana principal de la aplicación y contiene la ventana de vista, además de que puede contener otras ventanas como

la barra de herramientas, la barra de estado y barras de diálogo. Esta ventana se crea en la fase de diseño de la aplicación con *AppWizard*.

La clase que contiene este tipo de ventanas es *CFrameWnd*, de la que se derivan las clases *CMDIChildWnd*, *CMDIFrameWnd*, *CMiniFrameWnd* y *COleIPFrameWnd*.

La clase *CMDIFrameWnd* permite crear una ventana frame MDI para permitiendo de ese modo tener varias ventanas hijas (con sus vistas y documentos independientes). Además, la clase *CMDIChildWnd* debe ser el tipo de las ventanas hijas.

La clase *CMiniFrameWnd* es un tipo especial, usado normalmente en las barras de herramientas flotantes y que tiene la mitad de altura en la barra de título. Estas ventanas sólo tienen botón de cerrar y no disponen de menú del sistema.

En último lugar, la clase *COleIPFrameWnd*, nos permite editar otros objetos OLE en la misma ventana, cargando su menú y barra de herramientas, en vez de tener que abrir otra ventana.

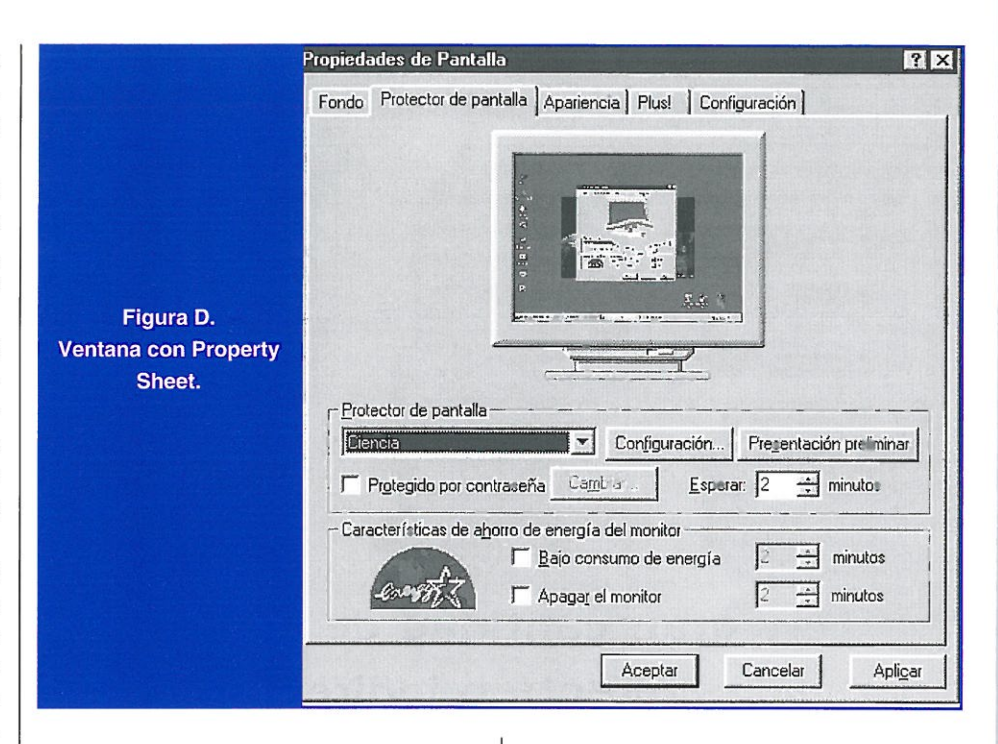


Figura D. Ventana con Property Sheet.

VIEWS

Las ventanas de vista o Views, se controlan desde la clase *CView*, permiten mostrar al usuario el contenido del documento cargado, así como permitir a éste que modifique dicho documento. Estas ventanas son hijas y están contenidas en la ventana/s frame de la aplicación.

Una vista sólo puede estar ligada a un documento, pero un documento puede tener varias vistas, ya sea mediante las ventanas splitter o bien cargando el documento en varias ventanas hijas MDI.

La ventana de vista es la responsable de manejar diferentes tipos de entrada, como el teclado, el ratón o incluso mensajes de drag and drop. De este modo se atiende a las acciones realizadas por el usuario.

El método normal para utilizar esta clase es derivar nuestra propia clase de ella, y sobrecargar la función miembro *OnDraw*, la cual nos permite pintar en pantalla los contenidos del documento.

La clase *CView* tiene además otras clases derivadas más especializadas como son *CEditView* basado en el control estándar de edición, *CFormView* basado en una caja de diálogo, *CListView* basado en el control lista, *CRecordView* para mostrar los registros de una base de datos en una caja de diálogo, *CScrollView* que permite atender de forma automática los mensajes de scroll, *CTreeView* basado en el control tree o *CRichTextEditView* basado en el control para edición de textos enriquecidos (RTF). De este modo y observando los posibles coincidencias con

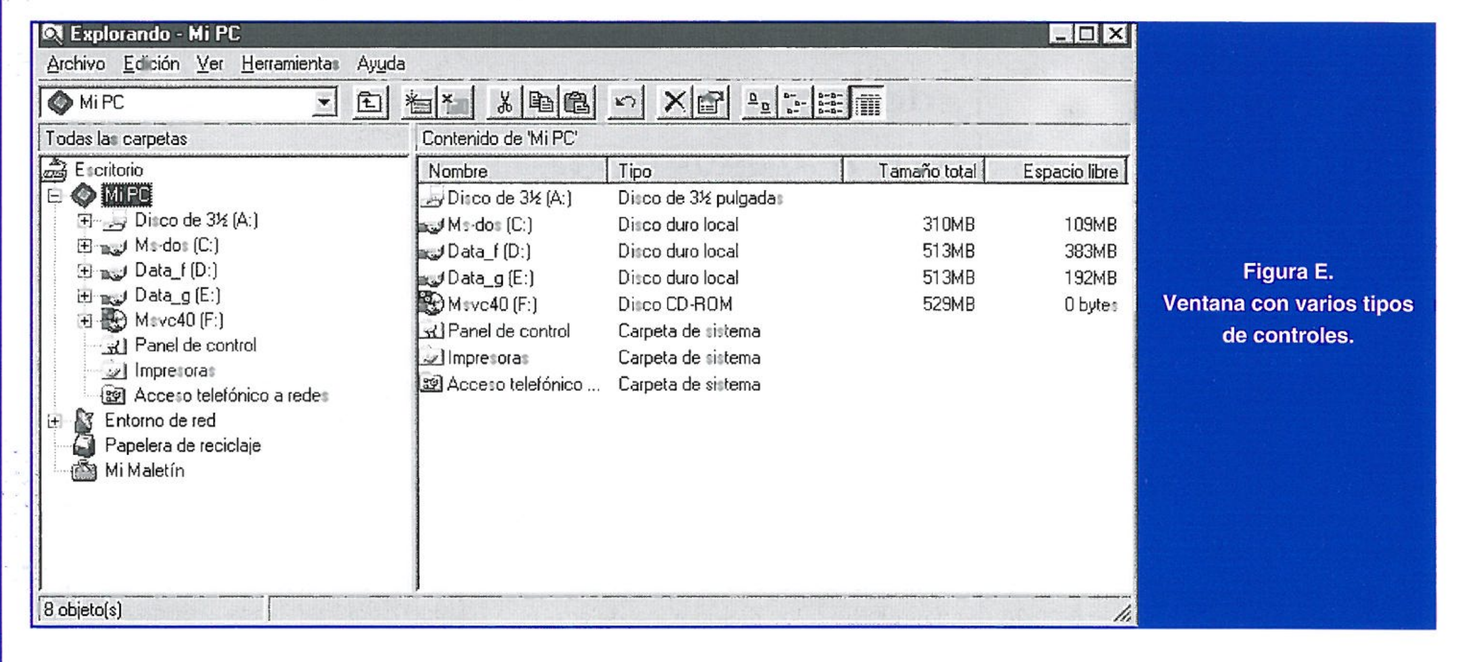


Figura E. Ventana con varios tipos de controles.

nuestra propia vista, derivaremos nuestra clase de una de ellas y particularizaremos nuestros propios métodos.

CONTROLS BARS

La clase *CControlBar* es una clase que encapsula un tipo de ventanas alineadas con las esquina superior izquierda del area de cliente de la clase padre. Estas ventanas suelen contener otras ventanas hijas (controles) que permiten generar o atender a mensajes Windows, lo que se traduce en métodos rápidos de acceso a diversas operaciones, como son las barras de heramientas o las barras de estado.

Las clases derivadas de esta clase son *CStatusBar* que muestra el estado de la aplicación en todo momento y que suele estar en la parte inferior de la ventana, *CToolBar* que encapsula los métodos de las barra de herramientas, *CDialogBar* que tiene la misma funcionalidad que una caja de diálogo modeless y puede ser diseñado desde el editor de recursos. Además puede alinearse arriba, abajo, a la derecha o a la izquierda. Si un botón de esta clase no tiene el manejador del mensaje *COMMAND* o *UPDATE_COMMAND_UI*, dicho botón es automáticamente deshabilitado.

PROPERTY SHEETS

La clase *CPropertySheet* representa este tipo de ventanas, también conocidas como tab dialog boxes. Este tipo de ventanas consisten en una colección de uno o varios objetos del tipo *CPropertyPage*, clase derivada de CDialog y que forman cada una de las páginas del tabulador.

CPropertySheet no es una clase derivada de *CDialog*, sin embargo el manejo de esta clase es similar al de una caja de diálogo. Al igual que en estas últimas, se pueden crear *Property Sheet* modales o modeless, mediante DoModal o Create respectivamente.

Además de tener el aspecto de los diálogos con tabs, esta clase también permite crear los conocidos *Wizards*, que son secuencias de diálogos que permiten interactuar con el usuario indicándole todos los pasos a seguir. Para ello basta con poner el *Property Sheet* en modo *Wizard* con la función miembro *SetWizardMode*.

Hay gran cantidad de clases derivadas directa o indirectamente de CWnd, divididas en varios grupos según su funcionalidad

DIALOG BOXES

CDialog encapsula uno de los tipos de ventanas más utilizadas en Windows, los diálogos. Estas ventanas son creadas en base a una plantilla, creada desde un editor de recursos, y donde visualmente se pueden colocar todos los controles necesarios para su manejo. Este tipo de ventanas se utiliza para transferir información a/desde el usuario de forma muy directa y mediante los controles contenidos en el diálogo.

Las ventanas de vista o Views, que se controlan desde la clase CView, permiten mostrar al usuario el contenido del documento

Existen dos tipos de diálogos en función del modo en el que hayan sido creados. El primer tipo, los diálogos modales, cuando se abre el diálogo, el programa principal suspende su trabajo hasta que el diálogo no es cerrado. El segundo tipo, los diálogos modeless, la

ejecución de la aplicación no se detiene mientras que el diálogo está abierto. Aunque el tipo segundo parece mejor por la oportunidad de continuar procesando en segundo plano, aveces es justo esto lo que se quiere evitar, y el tipo de diálogos modeless tiene alguna pequeña dificultad añadida en la programación.

CONTROLS

Los controles son quizás las ventanas más especializadas de todas. Cada uno de ellos tiene un labor muy bien definida, y que permiten desde actuar como un botón hasta realizar un editor con formato enriquecido. En las siguientes líneas vamos a realizar una breve descripción de los diversos controles, con el fin de tener una visión general de todos ellos y saber cual hay que utilizar.

CANIMATECTRL

Esta clase permite animar una secuencia de bitmaps, creando la sensación de movimiento.

CBUTTON

Esta clase encapsula la funcionalidad del control botón de Windows. Existen varios tipos de botones, los pulsables, las botones de selección y los botones circulares.

CDialog encapsula uno de los tipos de ventanas más utilizadas en Windows, los diálogos

CCOMBOBOX

Esta clase encapsula la funcionalidad del control *ComboBox* de Windows, que consta de un control lista y de un control de edición. Permite seleccionar un elemento de la lista en la caja de edición o bien insertarlo normalmente.

CEDIT

Es uno de los controles más potentes de los existentes en las versiones anteriores de Windows. Permite crear una ventana para insertar texto desde teclado como si de un editor sencillo se tratase. Se puede configurar para editar una sola línea o múltiples, permitir la escritura o no, e incluso ponerlo en modo password, por lo que en pantalla sólo aparecen asteriscos.

CHEADERCTRL

Es un control que se sitúa encima de columnas de texto o números y que permite ponerles un título y modificar el ancho de las columnas.

CHOTKEYCTRL

Es un tipo de control que permite crear una serie de teclas rápidas para realizar una determinada acción.

CLISTBOX

Es un control que muestra una serie de opciones, las cuales pueden ser elegidas por el usuario. Existen varios formatos de estas listas que permiten entre otras cosas mostrar los contenidos ordenados o permitir selecciones múltiples.

CLISTCTRL

Este permite mostrar una lista de iconos en pantalla. Se pueden mostrar de modo desordenado, o bien ordenado por columnas o por filas, y su uso más frecuente se encuentra en las visión que Windows 95 ofrece de sus carpetas.

COLECONTROL

Es una clase encargada de dar soporte a controles OLE, derivada de *CWnd*, con funcionalidad OLE añadida que permite lanzar eventos y soporta métodos y propiedades de los objetos.

CPROGRESSCTRL

Es un control encargado de informar de modo visual del estado, en tiempo o duración, de un determinado proceso, normalmente de una duración larga.

CRICHEDITCTRL

Esta clase encapsula uno de los controles más potentes, ya que amplía de forma notable la funcionalidad del control Edit, permitiendo disponer de varios tipos de letras, color, opciones de alineamiento, sub/superíndices,...

CSCROLLBAR

Esta clase encapsula la funcionalidad soportada por las barras de scroll de Windows.

CSLIDERCTRL

Es un control que se asemeja a un potenciómetro lineal. Tiene una barra con opción de contener marcas y un cursor que puede ser desplazado a lo largo de la barra.

CSPINBUTTONCTRL

Este control consta de dos botones en forma de flecha que permiten incrementar/decrementar el valor de un contador.

CSTATIC

Este control permite mostrar un texto, un rectángulo, un icono, un cursor, un bitmap o un enhanced metafile. Este control normalmente no recibe ni envía mensajes.

CTOOLTIPCTRL

Este control permite mostrar una única línea para indicar la función de un botón en la barra de herramientas.

CTREECTRL

Este control permite mostrar listas jerárquicas de elementos, en forma de árbol. Cada elemento se compone de una texto y opcionalmente de un bitmap. Cada elemento puede a su vez ser origen de otros.

CONCLUSIÓN

En este artículo se ha pretendido acercar al lector la jerarquía existente dentro de la familia de clases derivadas de *CWnd*, para saber en todo momento que clase debe ser base de nuestra propia ventana. Además se ha descrito el método general para crear un objeto *CWnd* y su ventana asociada, mediante el ejemplo del *beep button*.

En los siguientes artículos se va a entrar en los detalles de algunas de las ventanas ahora descritas por encima, lo que nos permitirá ir añadiendo nuevas funciones a nuestras aplicaciones, añadir controles y mejorar el aspecto general de las mismas.

CONTENIDO DEL CD-ROM

JDK 1.0.2: JAVA DEVELOPERS KIT de SUN MICROSYSTEMS

Este mes el habitual CD-ROM que acompaña a la revista contiene el JDK, Java Developers Kit, de SUN Microsystems en su versión para Windows 95 y Windows NT. En un futuro CD-ROM se incluirá un port que Randy Chapman está realizando para Linux. Asimismo, se ha incluido una nueva versión de la Biblia del Linux facilitada hace dos números que ha sido modificada para su visualización desde cualquier sistema operativo como DOS, Windows 95 o Linux. Para su visualización bastará abrir el fichero *top.htm* con un visor de WWW:

El JDK incluye:
- *javac*: compilador de Java
- *java*: intérprete de Java
- *jdb*: Depurador de Java
- *appletviewer*: Visor de applets de Java

Además, se han añadido otros ficheros como:
- Documentación de la API de applets
- Guía de programación de Java

Las versiones actualizadas de todos estos ficheros pueden encontrarse en el servidor de JavaSoft, *http://www.javasoft.com*

Dado que los ficheros zip contienen ficheros con nombres largos, para su extracción será necesario un descompresor que pueda emplear el formato de nombres largos de Windows 95. Para ello, en el mismo directorio Java se facilita la versión shareware del *WinZip versión 6.0.*

REQUISITOS

El JDK permite desarrollar applets que se ajustan a la versión final de la API. Ésta ha sufrido algunos cambios desde la versión 1.0 que figuran en la documentación.

INSTALACIÓN

Para proceder a la instalación del JDK es necesario seguir los siguientes pasos:
1. Borrar las versiones previas del JDK (si existen)
2. Descomprimir el fichero
3. Actualizar las variables de entorno

1. Borrado de versiones previas

El primer paso a seguir es el borrado completo de cualquier versión previa del JDK o de Java Workshop. En caso de que tenga almacenadas fuentes de programas en los directorios del JDK deberá moverlas previamente para salvaguardarlas.

2. Extracción del JDK

Un gran número de ficheros del JDK dispone de nombres largos no compatibles con las versiones clásicas de pkzip o winzip. Por ello, y en prevención de que el usuario no se haya actualizado a las versiones correspondientes, se facilita el kit en un fichero comprimido autoextraíble que recibe el nombre de *jdk102.exe*. En cualquier caso, se recuerda que para la extracción de otros ficheros como la documentación se facilita en el CD-ROM el WinZip para Windows 95. El JDK puede extraerse en cualquier directorio o subdirectorio bajo el cual creará un directorio padre denominado "java" y bajo él, todos los subdirectorios necesarios.

Durante la extracción se generarán dos ficheros con extensión *.zip*: *src.zip* y *lib/classes.zip*. El primero de ellos contiene las fuentes de algunas de las clases del JDK y se facilita por si el usuario desea revisarlas para ver cómo están programadas, pero no es necesario para programar. Este fichero también contiene ficheros con nombres largos. El segundo fichero debe permanecer comprimido. No se debe descomprimir el fichero *lib/classes.zip* bajo ningún concepto.

3. Actualización de variables de entorno

Para trabajar con el JDK es necesario añadir el directorio *java\bin* al PATH (o el directorio en el que se esté instalando). También es necesario definir una variable de entorno llamada *CLASSPATH* que debe apuntar al fichero *java\lib\classes.zip*. Para ello, suponiendo que el JDK se ha extraído desde la raíz del disco C: sería suficiente añadir las siguientes líneas al fichero AUTOEXEC.BAT:

PATH=%PATH%;C:\JAVA\BIN
CLASSPATH=C:\JAVA\LIB\CLASSES.ZIP

Si trabaja en Windows NT, los cambios en las variables se pueden realizar desde el submenú Sistema del Panel de Control.

EL COMPILADOR

El compilador de Java recibe el nombre de *javac* y admite la sintaxis:

javac [opciones] <fichero>.java ...
java_g [opciones] <fichero>.java ...

TABLA

Nombre	Nombre en internet	Descripción
api100.zip	JDK-1_0-apidocs.zip	Documentación para la versión 1.0
api102.zip	JDK-1_0_2-apidocs.zip	Documentación para la versión 1.0.2
tuthtml.zip	tutorial.html.zip	Tutorial de Java
jdk102.exe	JDK_1_0_2-win32-x86.exe	JDK versión 1.0.2
crigth.txt	COPYRIGHT	Copyright de SUN

Ficheros del directorio de Java.

El compilador toma un fichero fuente con extensión *.java* y proporciona como resultado, en el mismo directorio, un fichero de byte-codes independiente de la arquitectura con extensión *.class*. Para ejecutar este programa se podrá emplear en interprete *java*.

Si se definen clases propias será necesario especificar dónde se encuentran para lo que existe la variable de entorno *CLASSPATH*, en la que se indican los directorios adecuados separados por punto y coma. Si el compilador no encuentra una clase referenciada en los ficheros fuente que se le facilitan, buscará en los directorios definidos por esta variable.

El compilador *java_g* produce código no optimizado pensada para utilizarse con el depurador *jdb*.

OPCIONES DE COMPILACIÓN

El compilador javac admite las siguientes opciones:

-classpath <path>: Indica los paths de búsqueda de clases ignorando, si existe, el valor de la variable CLASSPATH.

-d directorio: Especifica el directorio en que se generarán los ficheros clase resultado de la compilación.

-g: Genera código con información de depuración.

-nowarn: Elimina los mensajes de warning.

-O: Genera código optimizado.

-verbose: Muestra información sobre las acciones que van realizando en compilador y el montador

EL INTÉRPRETE

El JDK facilita una máquina virtual Java con la que ejecutar ficheros de byte-codes y a la que se puede llamar con la sintaxis:

java [opciones] <nombre_clase> <parámetros>

java_g [opciones] <nombre_clase> <parámetros>

Los parámetros que se le pasen al intérprete pasarán al método *main()* de la clase. Este método debe estar definido de la forma:

```
class Ejemplo {
  public static void main(String argv[]) {
    ...
  }
}
```

El intérprete realiza un proceso de verificación de los byte-codes de cada clase para determinar si ésta es legítima. Al igual que ocurría con el compilador, *java_g* es una versión no optimizada pensada para ser utilizada con el depurador *jdb*.

OPCIONES PARA EL INTÉRPRETE

EL intérprete de Java es la aplicación que más opciones admite:

-debug: Permite que el depurador de Java, *jdb*, se incorpore a la sesión. Java mostrará un password que será necesario para iniciar la sesión de depuración.

-cs, -checksource: Actúa de modo similar al programa *make*. Al cargar una clase compara la fecha de última modificación del fichero de byte-codes con la de la fuente correspondiente y si procede recompila y carga la nueva versión. Java repetirá el proceso hasta que todas las clases se haya compilado y cargado correctamente.

-classpath <path>: Funciona del mismo modo que en el compilador.

-mx <tamaño>: Ajusta el máximo tamaño de de pila del recogedor de basura a "tamaño". El tamaño se especifica en bytes, en Ks o en Mbs añadiendo a la cantidad el carácter k ó m en los dos últimos casos respectivamente. Por defecto se toman 16Mb de memoria. El tamaño debe ser superior a 1000 bytes.

-noasyncgc: Desactiva al modo de recolección de basura asíncrona. Si se indica, no se produce recolección de basura a no ser que el programa lo indique expresamente o se quede sin memoria. Si no se indica, el recolector se ejecutará en un thread en paralelo con el resto del programa.

-ss <tamaño>: Cada thread en java accede a dos pilas, una para código Java y otra para código C. La opción ss ajusta el tamaño de esta última pila. La forma de indicar el tamaño es idéntica a la de la opción mx. El tamaño por defecto es 128k.

-oss <tamaño>: Ajusta el tamaño máximo de la cola para java (ver -ss). EL tamaño por defecto es 400k.

-v, -verbose: Muestra un mensaje cada vez que se carga un fichero clase.

-verify: Ejecuta el verificador sobre la totalidad del código.

-verify remote: Ejecuta el verificador sobre todo el código que se carga en el sistema a través de un cargador de clases (opción por defecto).

-noverify: Desactiva la verificación.

-verbosegc: Obliga al recolector de basura a mostrar una traza cada vez que libera memoria.

-Dpropiedad=<valor>: Permite redefinir el valor dela propiedad definida como propiedad asignándole como nuevo valor valor. Por ejemplo:

java -Dawt.button.color=green ...

Además de estas opciones, la versión *java_g* admite una más:

-t: Va mostrando una traza de todas las instrucciones según se ejecutan.

DOCUMENTOS HTML CON EXTENSIONES JAVA

Los documentos Html que incluyen directices Java son visibles solamente mediante un Browser Java compatible (Hotjava, Netscape 2.0) por lo tanto aquellos lectores que no dispongan de dicha herramienta deberán utilizar el visor *appletviewer*. Ejemplo: si se descomprimió jdk102. exe en el directorio raíz, entonces:

En el directorio Java/bin/demo/ blink para visualizar el fichero exampl~1.htm deberá teclearse;

appletviewer example1.html

NOTAS SOBRE COPYRIGHT Y LICENCIA

La distribución 1.0.2. del JDK realizada por Sólo Programadores se acoge a la licencia de SUN Microsystems que figura en el fichero COPYRIGHT de la distribución por imperativo en la cláusula que se detalla a continuación. Es obligación del lector leer el fichero:

"*SUN grants to you ("Licensee") a non-exclusive, non-transferable license to use the Java binary code version (hereafter,"Binary Software") without fee. Licensee may distribute the Binary Software to third parties provided that the copyright notice and this statement appear on all copies. Licensee agrees that the copyright notice and this statement will appear on all copies of the software, packaging, and documentation or portions thereof.*"



En esta sección, los lectores de SÓLO PROGRAMADORES tienen la oportunidad de hallar respuesta a los problemas que puedan tener en cualquier tema relacionado con la programación.

P *Hola y saludos a toda la redacción de esta estupenda revista. Estoy estudiando informática y quiero hacer un sencillo programa para plasmar y animar formas geométricas. Lo tengo en fase de diseño y todavía me falta alguna información. ¿Podrían explicarme la forma de dibujar circunferencias en pantalla, por supuesto de cualquier tamaño y posición? Gracias y a seguir así.*

José Luis Godás
(Cáceres)

R Sin recurrir a técnicas orientadas a 3D, lo más sencillo es utilizar el algoritmo de Bresenhams. Este algoritmo calcula la posición de los pixels tan solo para los primeros 45 grados, del total de los 360 que tiene la circunferencia completa.
Con esto es suficiente, pues las circunferencias son totalmente simétricas y con un octavo de su desarrollo se extrapola el resto.
El algoritmo quedará de la siguiente forma :

```
Circulo ()
{
d = 3 - (2 * RADIO)
x = 0
y = RADIO
REM Se repite esto hasta que x sea igual a y
while (x != y)
{
REM Se dibujan los pixels correspondientes
REM de los 8 cuadrantes
Pixel (CentroX + x, CentroY + y)
Pixel (CentroX - x, CentroY + y)
Pixel (CentroX + x, CentroY - y)
Pixel (CentroX - x, CentroY - y)
Pixel (CentroX + y, CentroY + x)
Pixel (CentroX - y, CentroY + x)
Pixel (CentroX + y, CentroY - x)
Pixel (CentroX - y, CentroY - x)

if ( d < 0 )
{
        d = d + (4 * x) + 6
}
else
{
        d = d + 4 * (x - y) + 10
        y = y - 1
}
x = x + 1
}
}
```

P *Me he animado a escribiros pues estoy harto de que mis programas se cuelguen porque los ejecutan en ordenadores un poco anticuados, que no soportan tal o cual instrucción o direccionamiento de memoria. A ver si me pueden indicar algún método sencillo y no muy extenso para detectar en tiempo de ejecución el tipo de microprocesador sobre el cual está corriendo mi software. Gracias y buenas compilaciones para todos.*

Arthur Paracol
(Madrid)

R Existen bastantes formas de detectar el tipo de procesador que está ejecutando al programa. Suponemos que se refiere a una plataforma a base de PC + Intel, que suele ser lo más habitual, y en donde más cambios ha habido en los últimos años.

Quizás la más sencilla sea la de provocar errores o "traps" controlados al intentar ejecutar alguna instrucción inexistente o no permitida en ciertos micros. Todos los chips posteriores al 80186 (incluido ese modelo), generan una interrupción 6 cuando se encuentran alguna instrucción que no saben como manejar ; de esta manera al intentar ejecutar instrucciones que se incluyeron a partir de ciertas versiones del micro, se conocerá si se trata de ese tipo o superior.

En la siguiente tabla se listan instrucciones que no existían en versiones previas del chip :

Instrucción	Soportada por :
shl dx, 5	80186 y superiores
smsw dx	80286 y superiores
mov edx, cr0	80386 y superiores
xadd dx, dx	80486 y superiores
cpuid	Pentium y superiores

Claro que antes de nada hay que comprobar que no se trate de un 8086 o 8088, pues el método anterior produciría un "crash" en el sistema. Esto se hará guardando el registro SP en la pila (push sp) y sacándolo a otro registro (pop ax), si ax y sp no son iguales, seguro que se trata de un 8086/8, pues estos chips incrementaban el valor del stack pointer (sp) antes de guardar algo, mientras que el resto de la familia lo incrementa después.

Para poder "cazar" la interrupción 6, se debe modificar el valor del IP

(Instruction Pointer) que se almacenó en la pila al generarse la interrupción. Dicho valor es la dirección del programa en donde se generó la excepción y si no se modifica, el procesador volverá a saltar ahí en un bucle sin fin. Todas las instrucciones de la tabla son de tres bytes de longitud, menos cpuid, que ocupa tan sólo dos ; por lo que habrá que sumarle 3 (o 2) al valor del IP de la pila.

P *Cuando desarrollo programas tanto en C como en Ensamblador y me encuentro con problemas del tipo "Desbordamiento de Pila" o que la pila se me descompensa, realmente no entiendo muy bien que es lo que está pasando, pues si compruebo el valor del SP (Sack Pointer), veo que no es muy elevado. Brevemente, ¿Como funciona la pila ?*

Rubén Nielpha
(El Escorial / Madrid)

R La pila es, literalmente, un área en donde "apilar" datos o valores. Y funciona como un montón de libros, en el que, el último que se depositó, será el primero en cogerse. Esto se denomina LIFO (Last In, First Out), o para los cristianos : el último en entrar, será el primero en salir. La única diferencia con el montón de libros es que la pila crece hacia abajo, comenzando desde arriba (pegando libros adhesivos desde el techo...). Por todo esto es facil averiguar que con cada Push, se decrementa la pila ; y con cada Pop, se incrementa. Si la pila va disminuyendo (mediante instrucciones Push) hasta bajar por debajo de 0, se habrá "desbordado".

Todo esto se ha de tener en cuenta al guardar y extraer datos de la pila, pues como se trata de un buffer LIFO, habrá que trabajar de la siguiente forma :

```
push ax
push bx
...
pop bx
pop ax
```

NO de la siguiente :

```
push ax
push bx
...
pop ax
pop bx
```

pues se habrán invertido los valores de estos dos registros.

P *¿Que productos están disponibles para trabajar con bases de datos compatibles Dbase desde el lenguaje C ?*

Nicolás S. Sosa
(Pilar / Argentina)

R Por aquí se conoce principalmente el paquete CodeBase, que aglutina sencillez, potencia y compatibilidad con casi todos los compiladores importantes. De todas maneras seguro que existen librerías freeware perfectamente funcionales.

P *Me dedico a desarrollar aplicaciones multimedia mediante el SDK de Windows y el lenguaje C, y me gustaría incluir efectos de tránsito entre pantallas (bitmaps) a modo de "cortinillas". ¿Conocen si existen funciones ya realizadas para ello?*

Santiago Vacas Molina
(Alcobendas / Madrid)

R Librerías de libre distribución con sus fuentes, no debe ser fácil encontrarlas, pero sí que sabemos de la existencia de módulos para Visual Basic y para Visual C dedicados a ello. Lo mejor para localizar lo que hay, es buscar por Internet. En páginas dedicadas a la programación bajo Windows.

P *¿Saben algo de mantenimiento o actualización remota de software utilizando Internet? Sería muy útil para el tema de drivers y bases de datos.*

Alejandro Malbarez
(Buenos Aires / Argentina)

R Esto lo llevan utilizando en mayor o menor medida muchos fabricantes de software. Microsoft te manda su última versión de Internet Explorer cuando conectas con ellos la primera vez (¡y sin avisar!) y dicen las malas lenguas que incluso corrigen *bugs* de Windows sin que se de cuenta el usuario. De la misma manera, los administradores de servidores y de sistemas Unix, configuran "su" sistema para poder mantenerlo, actualizarlo e, incluso, reinicializarlo, desde cualquier punto del planeta...

P *Estoy planteandome meterme en el mundillo de los vídeo-juegos, ya que tengo conocimientos de C y algo de ensamblador. La duda que tengo es si es cierto que los juegos están programados en su mayoría en ensamblador, ya que me parece increible debido a la cantidad de líneas de código que se requería. Yo para pintar cuatro sprites en pantalla he hecho una rutina con más de 500 líneas y no quiero ni pensar si sigo por este camino.*

José Enrique Fernán
(Arganda/Madrid)

R Lo del Ensamblador y los vídeo-juegos es a día de hoy más un mito que una realidad, lo cierto es que de los juegos que están saliendo actualmente al mercado contienen aproximadamente un 90% en C y un ridículo 10 % en Ensamblador. La "culpa" de ésto la tienen los nuevos procesadores y la complejidad de programar código optimizado teniendo en cuenta que pueden ejecutar varias instrucciones a la vez según en que orden hayan sido introducidas. Los nuevos compiladores de C se encarga de hacer esto por nosotros. Desde aquí te animamos a que continues adelante.

